Performance and Availability Evaluation of the Blockchain Platform Hyperledger Fabric

Carlos Melo*, Felipe Oliveira*, Jamilson Dantas*, Jean Araujo†*,
Paulo Pereira*, Ronierison Maciel*, and Paulo Maciel*
*Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil
{casm3, fdo, jrd, prps, rsm4, prmm}@cin.ufpe.br*
†Universidade Federal do Agreste de Pernambuco, Garanhuns, Brazil
jean.teixeira@ufape.edu.br†

***Abstract*—Through the blockchain-as-a-service paradigm, one can provide the infrastructure required to host blockchain-based applications regarding performance and dependability-related attributes. Many works evaluated issues and mitigated them to reach a high throughput or better downtime and availability indexes. However, to the best of our acknowledgment, studies regarding both characteristics are yet to be performed. This paper presents a performance evaluation of a private infrastructure hosting a blockchain-based application. As we monitored the system, we noticed some increase in resource consumption that may be associated with software aging issues on the Hyperledger Fabric platform or its basic components. Also, the impact of this resource increment on the probability of the system being operational has been evaluated. When consumption issues were considered, one of the transaction types increased the RAM consumption by almost 80% in less than 3 hours, reducing the system availability to 98.17%. For scenarios without resource increment issues on the infrastructure, the availability reached 99.35%, with an annual downtime of 56.43 hours.**

***Index Terms*—Blockchain, Performance, Measurement, Software Aging, Availability.**

I. Introduction

Blockchain is a technology that aims to solve trust problems and inefficient utilization of computational resources [1]. Usually, the provided solutions are the result of being **a peer-to-peer distributed architecture**, as well as the shared ledger immutability that (*tries to*) guarantees that after a transaction is processed and stored in the system, it can no longer be changed [2].

Blockchain-based applications usually demand high computational power to operate. Therefore, some transactions require specific hardware, and some networks demand more energy to perform transaction validations than entire countries with a population equal to or higher than 45 million people[1]. As the required resources to perform transactions rise, some issues are being discovered even in a smaller and private blockchain network.

As far as we can tell, to store billions of lines of logs and information about millions of performed transactions, **a lot of storage space, CPU, and memory access operations are required** even if we are dealing with hash codes that represents an entire block full of transactions. Therefore, as we are talking about a peer-to-peer system, these directives apply to users that can also be service providers.

Blockchain-based applications, like any computer system or service, require that some performance [3] and dependability [4] attributes be met for the service provisioning process at high-quality parameters. A system that takes too much time to process transactions will soon be left aside by their current users. A system that cannot be accessed 24 hours a day, seven days a week, by other means when the user wants to perform a transaction will likely be forgotten in limbo.

This paper evaluates the performance of blockchain-based applications running over the Hyperledger Fabric platform. We performed several experiments in order to demonstrate both transaction throughput and latency. However, by monitoring the system, we could notice the growth in resource consumption, which raises even more as we increase the number of submitted transactions to the testbed environment. We mighty characterize this phenomenon as a resource leaking common under environments with software aging issues and may be directly associated with the blockchain environment or the Docker engine or containers itself. We present the impact of this phenomenon over the availability of a baseline environment through a Stochastic Petri Net (SPN) model. The key contributions of this work are the following:

1) A performance evaluation methodology for blockchain environments based on Hyperledger Fabric platform;
2) experimental results that demonstrate the detection and characterization of software aging issues on a Hyperledger Fabric-based application deployed over a private infrastructure;
3) a performability model that represents the impact of the aging issues over the system's general availability.

The remainder of this paper is as follows. Section II presents some related works and compares them to what we had proposed in this paper. Section III presents the same basic concepts about the Hyperledger Fabric platform, performance measurement, software aging, and system availability. Section IV presents a high-level view of the performed experiments and modeling activities. Section VI provides a case study that summarizes the obtained results. Section V provides the performability model used to estimate the impact of the aging-related issues over this metric. Finally, Section VII presents the

[1]https://www.bbc.com/news/technology-56012952

conclusions and future works.

II. Related Works

Over the last few years, authors have devoted their efforts to evaluating performance and dependability associated metrics of blockchain platforms and based applications. However, to the best of our knowledge, none of them has evaluated both characteristics of these systems simultaneously. Some of the closest related works to what is proposed by the current paper are listed below.

Pongnumkul et al. 2017 [5] proposed a performance evaluation methodology for blockchain-based environments and evaluated both Ethereum and Hyperledger Fabric's platforms. The authors concluded that the Hyperledger Fabric has a large throughput and lower latency than Ethereum, much like the consensus protocol adopted. While Ethereum uses a Proof-of-Work (PoW) protocol, the Hyperledger Fabric worked with the Practical Byzantine Fault Tolerance (PBFT) protocol.

In Thakkar et al. 2018 [6], the authors evaluated the performance of Hyperledger Fabric and identified some of its performance bottlenecks. The paper proposed improvements on platform infrastructure based on parameter variation, such as block size and endorsement policy over the transaction throughput and general latency. As a significant result that we may cite, the authors could improve the overall throughput by 16 times. Once again, the authors focused only on the system's performance issues, putting aside dependability attributes differing from our paper in both evaluation methods and metrics.

The PBFT protocol is one of the most used by blockchain platforms, nearly as important as the PoW protocol. Already in Sukhwani et al. 2017 [7], the authors investigated whether a consensus process using the PBFT mechanism could be a performance bottleneck. The authors evaluated the system through experimentation and Stochastic Reward Nets (SRN) modeling, which was later used to compute critical performance metrics, such as the meantime to consensus.

Later, in Sukhwani et al. 2018 [8], the authors evaluated the performance of the Hyperledger Fabric through SRN models, which means that this works dealt with the performance evaluation of the entire platform, improving their previous work that focused only on the consensus protocol. Once again, the proposed work differs from ours in the evaluated metrics, modeling formalism, and, most importantly, the main objectives. The authors used the SPNP package to construct and evaluate their models in both papers.

Melo et al. 2018 [9] evaluated the availability of the Hyperledger Composer environment through Dynamical Reliability Block Diagrams (DRBDs), which are an extension from traditional RBDs that includes the possibility to represent dependency between components. This paper was the first to model and evaluate the availability of a blockchain-based environment.

The authors in Melo et al. 2021 [10] evaluated the availability and deployment costs related to service provisioning of an Ethereum private blockchain environment. The authors also used Stochastic Petri Nets to model the evaluated environment. However, they do not perform any performance experiments.

This paper provides performability modeling and performance measurement to achieve real-world data. The main advantage of the proposed approach is that we may evaluate a set of larger scenarios derivated from the baseline one, while some other publications that focus only on performance measurement will work only for specific hardware and software. Table I presents the main differences between the related works and what we propose in this paper.

Publication	**Performance**	**Availability**	**Modeling**	**Measurement**
Pongnumkul et al. 2017 [5]	✓			✓
Thakkar et al. 2018 [6]	✓			✓
Sukhwani et al. 2017 [7]	✓		✓	✓
Sukhwani et al. 2018 [8]	✓		✓	✓
Melo et al. 2018 [9]		✓	✓	
Melo et al. 2021 [10]		✓	✓	
This paper	✓	✓	✓	✓

TABLE I: Main Contributions

III. Background

This section presents the fundamental concepts about the Hyperledger Fabric platform, performance measurement, software aging, availability evaluation, and system modeling.

A. Hyperledger Fabric

The Hyperledger Fabric[2] (HLF) is a platform for the construction and deployment of solutions based on shared ledgers. This platform is an open-source standard developed and maintained by the Hyperledger Consortium[3] under the tutelage of the Linux Foundation.

The service provider side on an HLF environment uses container-based technology on smart contracts (chain codes) management. Generally speaking, the chain codes are responsible for managing the business rules and how the application will perform its activities. These chain codes must be preinstalled on the environment as it is deployed. Already on the client-side, an SDK written in Node.JS or Java provides the communication between the server and client parts.

The Hyperledger Fabric's environment deals with three different types of containers:

1) The peer Node, which performs endorsement
2) The Membership Service Provider (MSP) Node, responsible for membership and access to the platform
3) The orderer Node that receives and assigns transactions to batches and send them back to the peer Node

This high-level view is essential since our models will be provided based on these components' relationships.

[2]Hyperledger Fabric: https://www.hyperledger.org/projects/fabric
[3]Hyperledger: https://www.hyperledger.org/about

Usually, a client uses its SDK to send a transaction to the service provider. However, many other steps are required and must be first accomplished on the other side to perform a transaction. This paper evaluates a basic application deployed over the three Hyperledger Fabric containers.

Figure 1 shows how the client and the service provider communicates, as well as which steps must be followed by a transaction in order to it be entirely accepted by the system [11], [12].

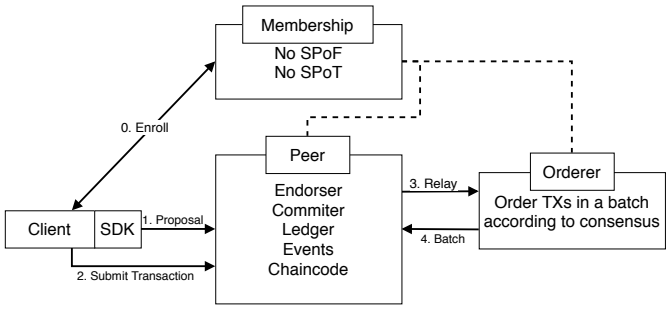


Fig. 1: Hyperledger Fabric's Overview

The client connects to the system through the **Membership** Service Provider (MSP), which should provide no Single Point of Failure (SPoF) and no Single Point of Truth (SPoT). The MSP verifies the credentials of this client and provides access to the service provider. Later, the client proposes a transaction to the Node known as Peer. There are many kinds of peers. Our focus is on endorsement peers, who simulate a transaction and send it back to the client with a signature, determining if the system can perform this transaction.

B. Performance Evaluation

Performance evaluation deals with the process of measuring and analyzing the performance of a particular system under test (SUT) [13]. Usually, it aims at the system's understanding and documentation based on specific situations, as well as on the administrator's needs, interests and skills [14].

Figure 2 presents a high-level view of the evaluation process adopted by this paper, where a blockchain network composed of *n*-Nodes is observed by a client and submitted to test by another.

The **Test Harness** corresponds to all software and hardware used in the benchmarking process by sending transactions and performing observations about the system under test (SUT) [15].

The **observer client** is the kind of Node that receives notifications about the SUT and the submitted transactions. Observing clients can also submit transactions to the system, which is performed by the Caliper benchmark, meaning that the observer client is also the **Load Generating Client**.

The **System Under Test (SUT)** corresponds to all specific hardware, software, networks, and configurations required to execute and maintain a Hyperledger Fabric based-blockchain [15]. While a **Node**, in the context of blockchains, is any independent virtual entity that communicates with other nodes in a network to perform transactions [15].

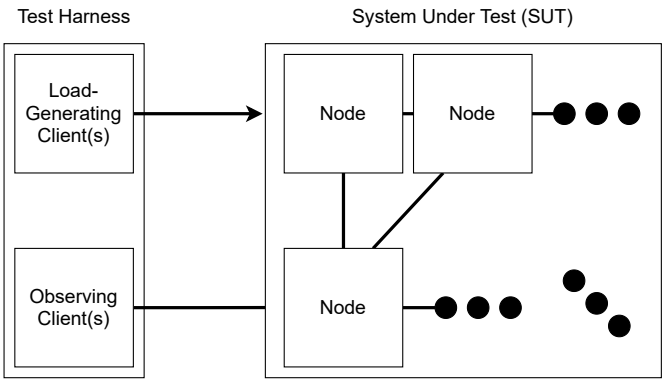


Fig. 2: Typical setting for a blockchain benchmarking (Based in [15])

The benchmarks are a set of standard tools used to compare one system to another or with a previous version of itself [14]. A specific type of benchmark is the performance benchmark, which can be designed as an informal implementation, based on scripting, manual testing, and even be a part of a set of globally established standards as the Standard Performance Evaluation Corporation (SPEC) [16].

Hyperledger Caliper [7] is a blockchain benchmarking tool developed by the Hyperledger consortium. Caliper allows its users to measure the performance of a specific implementation of a blockchain with a predefined set of case studies. Caliper's reports have performance indicators such as TPS, latency, and container resource utilization.

One of Caliper's key features is its adaptability, which allows the user to compare the performance of several different blockchain implementations and helps administrators define which one best suits their interests. In this paper, we evaluate the performance of Hyperledger Fabric through Hyperledger Caliper.

C. Software Aging

Software aging is a well-known phenomenon coined back at AT&T Bell Labs back in the 1990s [17], but aging-related issues have been around many years before. Three decades have passed since Bell Labs, and much still needs to be done regarding these issues on regular software, large systems, and new mobile applications, which are on a meteoric rise for the last ten years. Trends like Edge and Fog, which extend Cloud computing environments, Internet-of-Things, Big Data, Blockchain, and many other topics, also lack studies regarding the impact of aging issues over service provisioning.

Each component must be well thought out regarding software development, aiming to extend its use as much as possible. However, the demand for new applications and functionalities is rising; isolated human errors become more frequent and translated to code. As expected, now or in a moment in the future, these errors will be paid with a high-interest rate.

Software aging is the degradation of the internal state of software during its operational life [18]. Due to its cumulative property, the software aging occurs with more intensity at

systems that have been running for long periods [19]. Such systems also have a higher probability of entering into a critical condition, meaning that software failures can damage their integrity [20].

D. Availability

The system's availability is the probability that a system is operational at a time instant t [21]. Also, availability is the dependability attribute evaluated by this paper, divided into two subcategories: instantaneous and steady-state availability. The steady-state availability, which is the one evaluated by this paper, is also called the long-run availability and can be measured by the limit of 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 calculated through a ratio between the Mean Time To Failure (MTTF) and the Mean Time To Repair (MTTR) of the given system (Equation 2). While the MTTF refers to the expected time that a system will work before entering into a failure state, the MTTR is the expected time taken for a system into a failure state to be repaired [4].

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

The system's unavailability is the counterpart of the system's availability and can be calculated by $1-A$. The system's downtime is usually given in units of time, such as in hours per year, relating both time and probability: Unavailability $\times$ 8760 h.

E. Modeling

The system's availability of associated metrics and values can usually be evaluated through simulation, measurement, and models. The latter was chosen due to the high-level system view and their great flexibility in adapting parameters and achieving results faster than the other two evaluation techniques. This type of model can also be called a white box or structural model and enables us to create a mathematical representation of the system based on its relationship. Regarding modeling, we may highlight two main trends: Combinatorial and State-space models [22].

Reliability Block Diagram (RBD) [21] and Fault Tree (FT) [23] 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 formalism [21], [22], [24].

State-space models such as Petri Nets and it's extensions represent the system behavior based on the occurrence of events (rates or distribution functions). These models may represent the increase in resource consumption by software aging issues using timed related transitions.

This paper adopts Stochastic Petri Nets, an extension of Petri Nets that accommodates timed transitions to describe the relationship between resource leaking and the system's general availability. This approach has already been adopted and proved to be effective by some remarkable papers, such as Araujo et al. [25] [26], that used Petri Nets to estimate the impact of the resource consumption generated by aging issues over the failure of cloud computing infrastructures.

IV. Experimental and Modeling Methodology

Figure 3 presents the experimental and modeling methodology regarding the evaluated environment, which is divided into three main activities: (i) planning; (ii) deployment and measurement; (iii) performability modeling and evaluation.

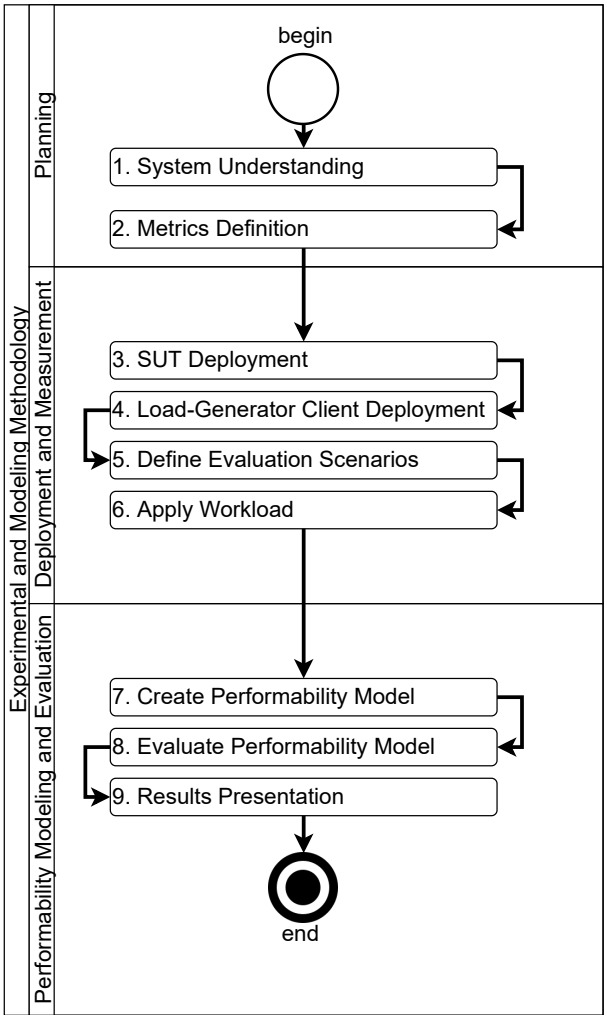


Fig. 3: Experimental and Modeling Methodology

The **planning** phase covers the first two steps of the modeling and evaluation methodology: (1) system understanding; (2) metrics definition.

The system understanding step is characterized by understanding how the Hyperledger Fabric works by identifying its main components and how they accomplish service provisioning. Later in the metrics definition step, we establish the metrics we are interested in for both the general system's performance and specific values for given containers.

The **deployment and measurement** phase covers four other steps related to performance evaluation: (3) SUT Deployment,

(4) Load generator deployment, (5) define evaluation scenario, and (6) apply the workload.

The SUT deployment step consists of deploying the operative requirements at the System Under Test environment (See Table III). The counterpart is the Load Generator Client Deployment (See Table II).

Load Generator Client	
OS	Ubuntu 18.04 LTS
PROCESSOR	Intel Core i5-3570K
MEMORY	8 GB
STORAGE	500 GB HDD
CALIPER	v0.2.0
NODE.JS	v10.24

TABLE II: Load Generation Client - Settings

System Under Test (SUT)	
OS	Ubuntu 18.04 LTS
PROCESSOR	Xeon E3-1220V3
MEMORY	32 GB
STORAGE	500 GB HDD
HYPERLEDGER FABRIC	v1.4.1
DOCKER ENGINE	v18.09
DOCKER COMPOSE	v3.7

TABLE III: System Under Test - Settings

Then we define the evaluation scenario, which means the workload that will be applied to the SUT in the later step. A set of scenarios were defined in this step, and they vary in the type of transactions submitted to the system and transactions sent by second. We develop new scenarios until we consider that we have covered many outcomes.

The third and last phase comprised three other steps and was attached to this methodology as Performability Modeling and Evaluation, the steps in this phase are respectively: (7) create performability model; (8) evaluate performability model; (9) result presentation. The first step is characterized by identifying the main system's components and moving them to an performability model through the Mercury modeling tool [27]. Then we evaluate this model and present the results as tables graphs. If something seems off or not satisfactory, we refine those models by going back to previous steps.

V. Performability Model

This section presents the proposed model that combines performance and availability evaluation (therefore, **performability model**) for Hyperledger Fabric's blockchain-based application. Figure 4 presents a high-level view of the modeled system through a service stack. It is important to highlight that the hardware and resources components on the base of **this stack indicates that the general computer hardware resources** (motherboard, CPU, Disk, RAM.) may fail given the passage of time (MTTF), while other components such as disk and RAM may also fail due to exhaustion, leading to a system failure as well.

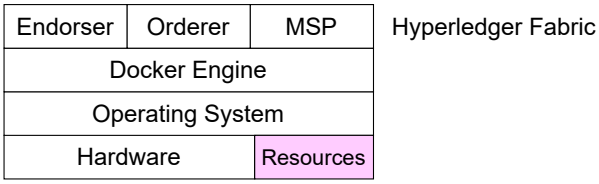


Fig. 4: Service Stack

To represent a Hyperledger Fabric-based architecture and the fact that its operational model demands that all components must be fully operational in order to accomplish service provisioning, we present our performability model in Figure 5. It is important to mention that this model is generalizable, and in order to represent a large architecture, it requires that the evaluator to add more tokens on the given transitions.

The first part of this model is under the set of the red rectangles, and from left to right represents the main components of a **Hyperledger Fabric basic environment**, which are respectively Hardware (**HW**), Operating System (**OS**), Docker Engine (**DE**), Endorser (**EN**), Orderer (**ORD**), and Membership Service Provider (**MSP**). **Initially, all these components are operational**, indicated by the black circles (tokens) inside their respective ***_ON places (white circles). **When any of these components fail**, through a transition called MTTF_*** is fired, **they enter into a failure state**, indicated by their counterpart or the ***_OFF places.

The second part of the model, under the green rectangle, deals with the system's current status. The presence of a token at **SERV_ON indicates that the service is operational**, and to accomplish it, it depends that all of the Hyperledger Components be on their respective ***_ON states and that at least one token to be hosted on the RES_FREE place.

Table IV presents the guards in both transitions COMP_FAIL and NO_RES, which are the transitions responsible for changing the service state. The COMP_FAIL immediate transition indicates the service provisioning failure. O**ther important places** are inside the blue rectangle, as the **RES_FREE** place, which deals with the number of available resources. The **NO_RES** transition is fired when there is no resource available. The NO_RES transition is fired, leading the server from SERV_ON place to the SERV_OFF.

Transition	**Guard Expression**
COMP_FAIL	(HW_ON=0)OR(OS_ON=0)OR(DE_ON=0)OR (EN_ON=0)OR(ORD_ON=0)OR(MSP_ON=0)
NO_RES	#RES_FREE=0

TABLE IV: Guard Expressions for the Performability Model

Another highlighted area is inside the pink rectangle. The **MAIN_TEAM's place indicates that at least a maintenance team is available** to repair the system through an MTTR represented by the large transition connected to both parts of our model. **When maintenance happens, all components into a failure state are repaired**, and a token is moved to the SERV_ON state. However, the maintenance team must be first called through a transition named CALL, which indicates a call from users or the administrative personal,

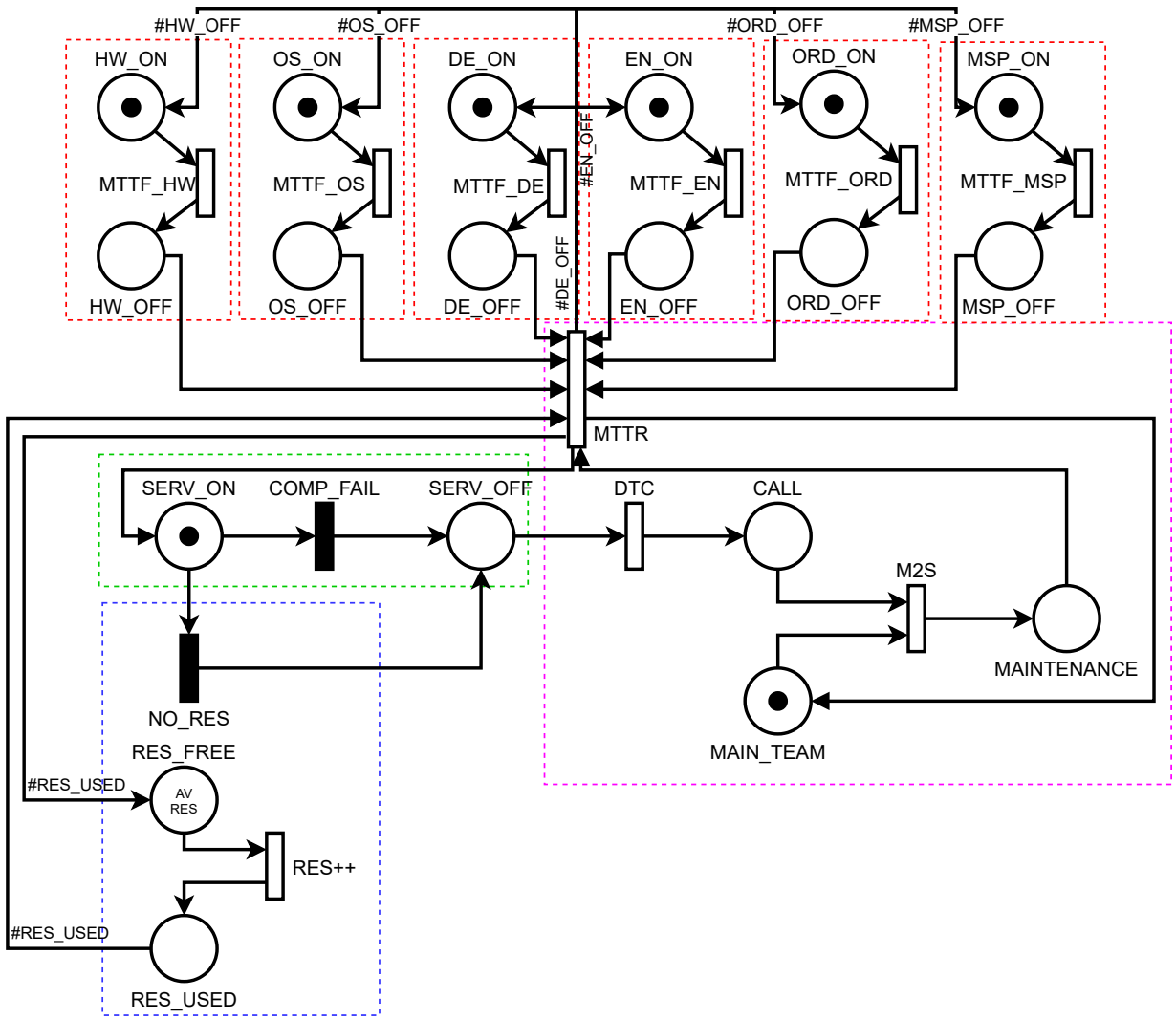


Fig. 5: Performability Model representing an environment with Resource Leaking

then the maintenance team takes some time to move to site (M2S) where the service is being hosted. Also, administrative time to detect the failure is required, and previous specified, represented by the detection (DTC) timed transition.

When in the presence of resource leaking issues, we may estimate how much of the leaking resources will be consumed by a unit of time (RES++), and as the RES_USED becomes too high, there is a great possibility that the system will enter into the failure state (SERV_OFF). This model tries to estimate the impact of the aging issue named resource leaking over the general system's availability.

VI. Experimental results and performance evaluation

This section provides the experimental results and the performance evaluation regarding a Hyperledger Fabric environment, followed by the resource leaking issues found and the availability evaluation.

A. Performance evaluation

The used benchmark is an example provided by the **Hyperledger caliper** developers that test the system's performance by running simple account transactions, such as opening new accounts, querying, and transferring from an account to another. Caliper's simple benchmark was chosen and performed over a Hyperledger Fabric v1.4.1 environment. Figure 6 presents an overview of the performed experiment.

Some additional information about the performed experiment is listed below. All those values were chosen in order to increase our control over the platform and were considered fair values for a long run experiment:

- We have defined a fixed amount of transactions by experiment run (ten thousand);
- We have fixed the transaction's send-rate;
- Each experiment was performed thirty times for each defined setting.

We performed the opening account transactions at the first level of the experimental analysis, later the queries' transactions, and finally, the transfer between accounts transactions

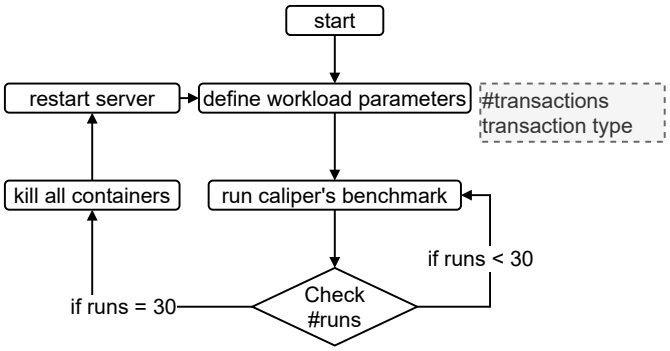


Fig. 6: Experimental Overview

from an account to another was performed. After each experiment run, the Hyperledger Fabric's containers were removed, and the SUT rebooted.

A total of 390 opening account transaction runs were performed. Every thirty rolls, we increased the sending rate by 10, starting from 30 rolls with a sending rate of 40 transactions per second (TPS) and reaching 30 rolls of a sending rate of 170 TPS. The Figures 7 and 8 present the mean opening account's transaction throughput (tps) and latency(s) respectively.

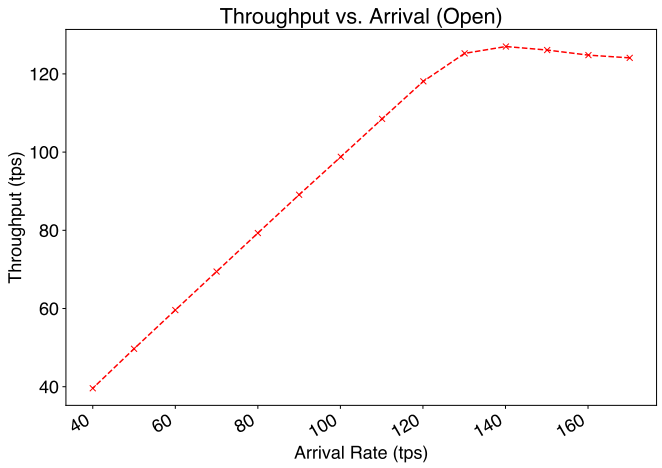


Fig. 7: Opening Account - Throughput(tps)

In the opening account scenario, as can be seen in Figure 7, the throughput grows as we increase the sending transaction rate until we reach approximately 130 transactions per second and then starts a decline, meaning that we reached the maximum processing capacity of our SUT regarding the opening account transaction, or by other means, our service queue is now full. This information can be confirmed by Figure 8 that presents the system's latency in seconds, which starts stable at ≈0.5 seconds and goes up more than 400% until it reaches ≈2.5 seconds. The scenario keeps stable until it reaches a send rate of 130 transactions per second, then the transactions start to demand more time to be performed, meaning that they start to be queued by the server and in a moment in the future will start to be discarded.

One hundred eighty runs were performed for the querying scenario, increasing the sending rate by 20 tps after 30 new rolls. The experimental study starts at 260 and goes up to 420

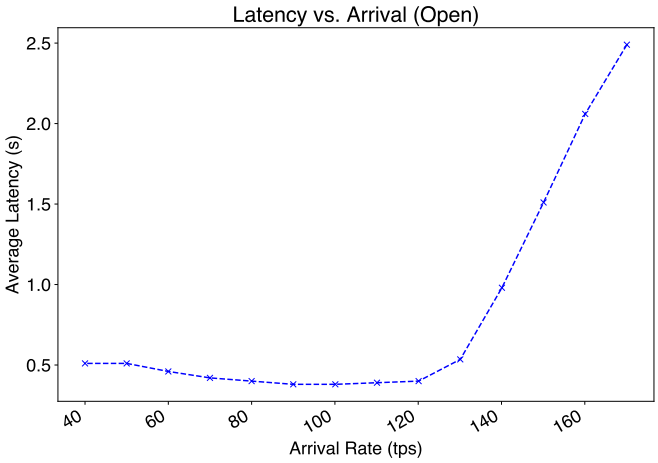


Fig. 8: Opening Account - Latency(s)

tps. The Figures 9 and 10 presents the mean querying transaction throughput(tps) and the system's general latency(s).

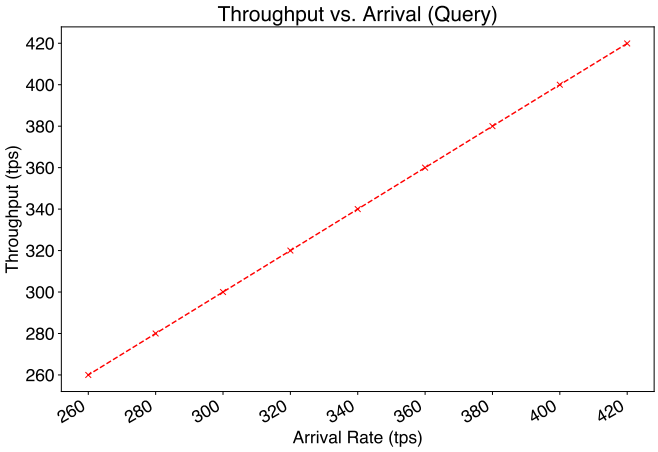


Fig. 9: Query - Throughput(tps)

Figure 9 shows that **the query's transaction throughput is high compared to the opening account transaction**. It grows up as we increase the sending rate and does not stop at the middle, as previously stated in the opening account transactions. **The query transactions have almost no impact on the performance of our SUT**. We started at a throughput of 260 transactions per second and reached 420 transactions per second with space to increase it even more. While Figure 10 presents a high increment (≈600%) in latency, but it is still too low if compared with the opening account transaction type, starting at a hundredth of second and going up to 0.07 seconds.

Last but even more important is the most performed transaction in general blockchain applications: transfers between accounts. This scenario is characterized by increasing the sending rate by ten tps every 30 rolls, starting at 40 and going up to 120 tps. We presented the obtained results for this scenario in Figures 11 and 12 representing respectively the transaction throughput(tps) and the system's general latency(s).

Figure 11 shows that like the opening account transaction

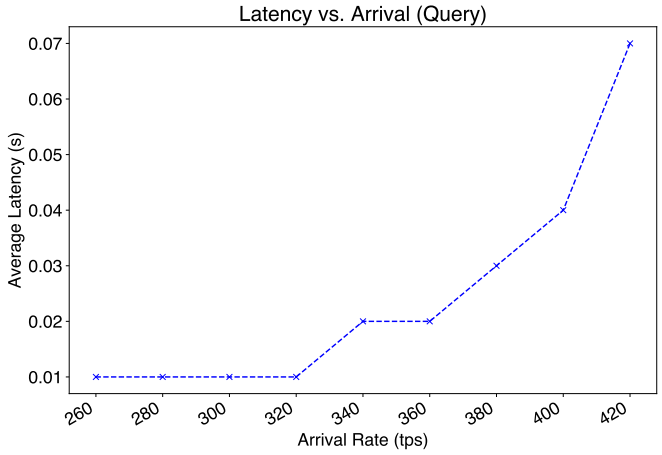


Fig. 10: Query - Latency(s)

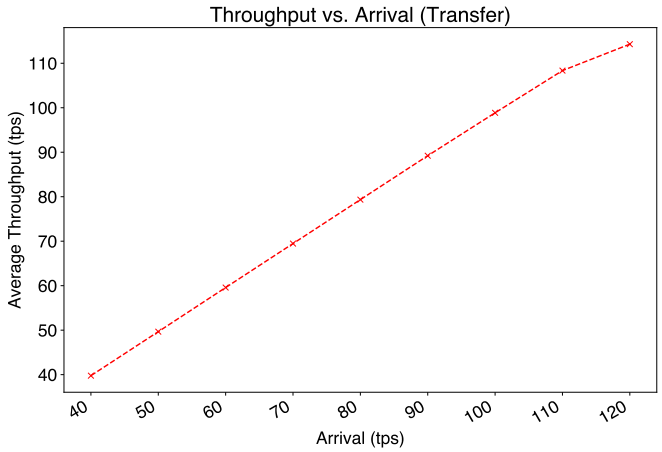


Fig. 11: Transfer - Throughput(tps)

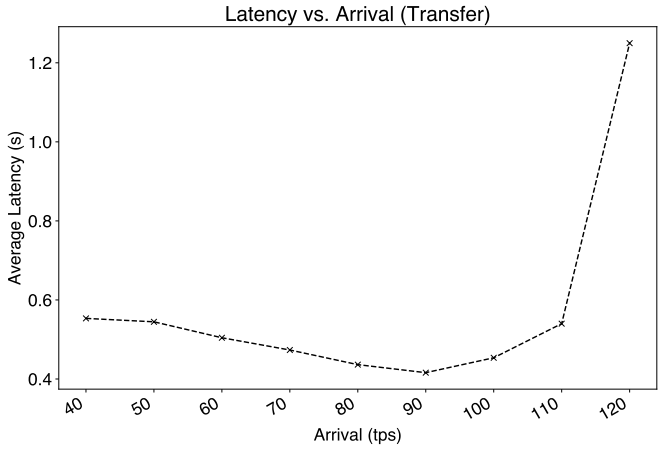


Fig. 12: Transfer - Latency(s)

type, the **transfer transaction type stagnates as we reach a high sending rate** (120+ tps). We started at 40 transactions per second sent and processed, then we went up to 120 transactions sent but only 115 processed by the system, meaning once again that our SUT is overloaded and our service provisioning queue is now full.

The **latency in seconds goes up as well** for transfer transactions, as we saw in Figure 12. Once again, we reach a threshold of our SUT regarding this type of transaction. The latency started at ≈0.5 seconds for 40 transactions sent by second. By the end of our measurement, the latency had increased more than 100% (Passing 1.2 seconds) when we improved the sent rate from 110 to 120 transactions per second. In the next case study, we present the resource's consumption for the evaluated transactions type and point out what may be a set of aging issues regarding leaking.

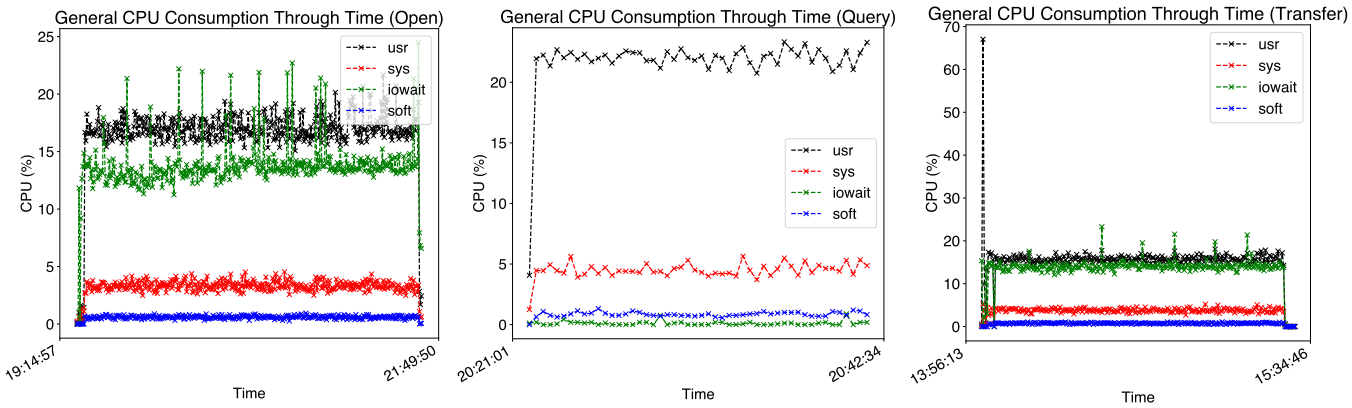
B. Resource Consumption

We have **monitored the system and available resources at many levels through bash scriptings** for CPU, RAM, Disk, and cached memory. **Here we present the status of the monitored resources at the highest sending rate** value supported by our SUT and presented in the previous case study. Software aging issues were observed in the evaluated SUT regarding Disk and RAM resource consumption.

The CPU is highly stressed at the *user* and *iowait* levels regarding opening accounts and transfer between accounts transactions. However, we did not reach 100% of CPU usage during our experimental study, meaning that the system's bottleneck for these transactions types might be another system's component, such as the disk or RAM. Already for the query transactions, only the user level provides a high consumption of CPU results while *io*, *soft* and *sys* keeps on a lower consumption level. Figure 13 shows the CPU resources consumed for each transaction type evaluated.

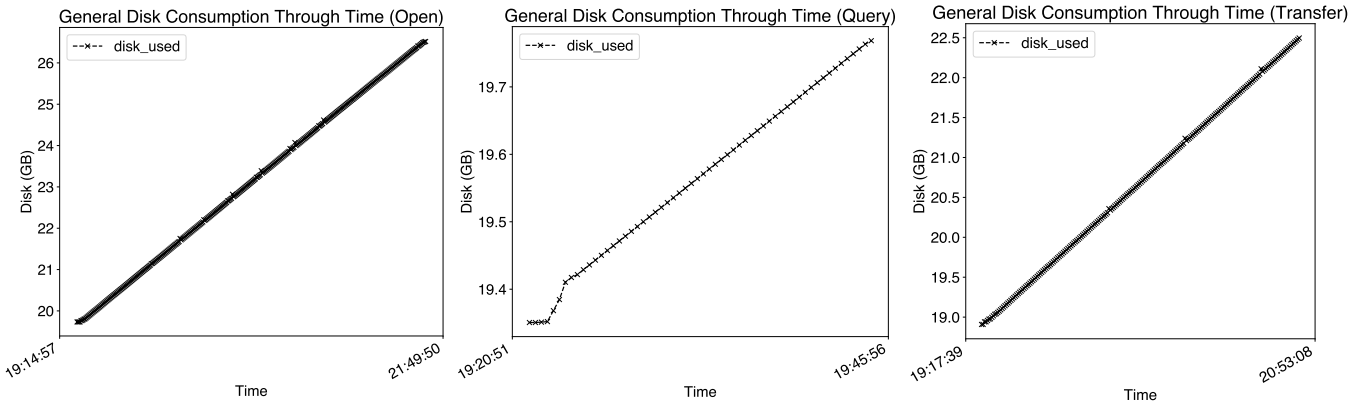
The disk resources' consumption increased, implying a high demand for too much time, impacting the system's performance. The disk exhaustion may lead to failures at the operating system and application levels. The highest impact over these resources is from the opening account transaction. In less than 2 hours and 35 minutes, more than 7 GB of the disk resource was allocated to the blockchain environment. **Transfer transactions had a high consumption regarding disk resources**, with almost 4.5 GB in less than 2 hours. Figure 14 provides the consumption of disk resources by each type of performed transaction.

As we dealt with HDDs, which are pretty slow compared to CPU and RAM, this component might be the system bottleneck regarding latency. As all transactions are persisted on the disk, some improvement might be made with an SDD or NVMe device. As an extension to the disk, **the cached memory increased by the same amount of the disk**, which is expected since the cache process allocates the disk's space to accomplish the required tasks. However, this new vision provides us with a start from zero to the peak. Figure 15 presents the increase in cached memory for each transaction type.



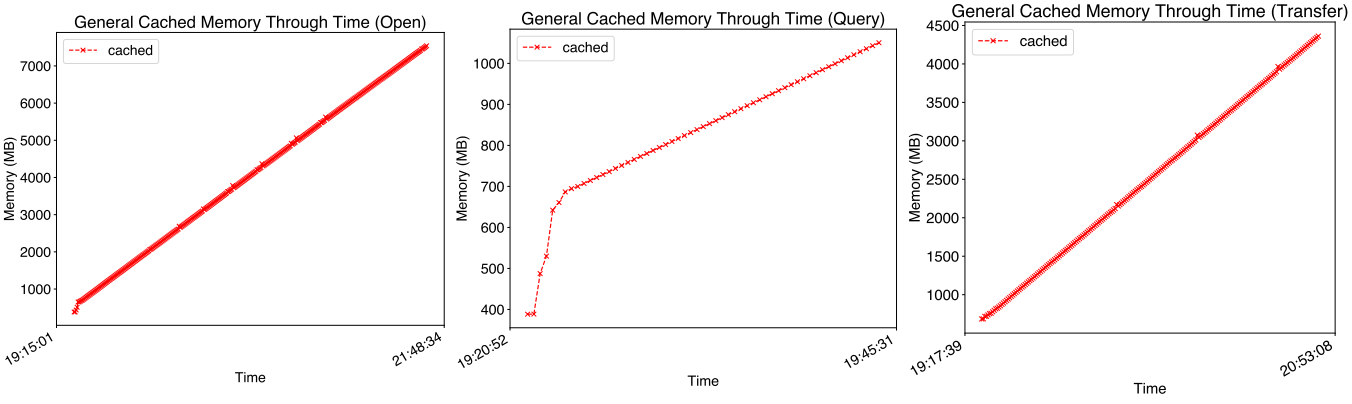
(a) Open Consumption - CPU (b) Query Consumption - CPU (c) Transfer Consumption - CPU

Fig. 13: CPU Consumption per Transaction Type



(a) Open Consumption - Disk (b) Query Consumption - Disk (c) Transfer Consumption - Disk

Fig. 14: Disk Consumption per Transaction Type



(a) Open Consumption - Cached (b) Query Consumption - Cached (c) Transfer Consumption - Cached

Fig. 15: Cached Resources per Transaction Type

The last highly consumed resource is the main memory. The highest impact on this resource is from the open account transaction type. For this transaction type, the RAM consumption that started at 800 MB increased up to 1.4 GB in less than three hours, meaning that **the RAM consumption increased by almost 80%**. The RAM consumption kept stable for query and transfer transactions after the experiment started. Figure 16 presents the RAM consumption for each type of transaction performed.

The indicative of software aging cannot be denied, mostly for long-running environments. In less than 3 hours at high throughput, we noticed the consumption level increment by the monitored general resources. These issues may impact performance metrics and dependability attributes such as the system's availability, presented below.

C. Availability Evaluation

The last evaluated metric was the system's availability and annual downtime regarding the presence or absence of resource leaking. We used the previous obtained data to **establish a mean of resource consumption per unit of time** (hour). The resources considered here are disk and RAM consumption.

In order to perform the availability evaluation of the proposed model, some input parameters are required; those were obtained from literature review [10], [28], [29], while other ones are from previous case studies. Table V present the input parameters used to evaluate the proposed models.

Transition	Time (h) or Condition
HW_MTTF	8760
OS_MTTF	2893
DE_MTTF	2516
EN_MTTF	1258
ORD_MTTF	1258
MSP_MTTF	1258
MTTR	1
DTC	0.5
M2S	0.5
COMP_FAIL	Failure of Any Component
RES++	Depends on the resource
NO_RES	RES_FREE = 0

TABLE V: Input Parameters for Availability Evaluation

The resource consumption metric considers a relationship of GB/h for both RAM and disk-related issues. This scenario considers an estimation that has as a basis the values obtained while monitoring opening, query, and transfer transactions. It is important to highlight that we have 500 GB of disk and 32 GB available, and the estimations that will be provided consider a clean environment where the current applications only use our resources. Table VI presents the current consumption per hour for the considered resources regarding the transaction type.

Based on the input parameters and consumption per hour presented in Tables V and VI **we could evaluate the proposed model**. Table VII provides the general availability obtained results with and without issues regarding resources consumption, where Av. stands for Availability and A. Down. is an acronym for Annual Downtime.

Scenario	Consumption per Hour
Opening Accounts - Disk	3 GB
Opening Accounts - RAM	0.24 GB
Query - Disk	2 GB
Query - RAM	0.1 GB
Transfer - Disk	1.67 GB
Transfer - RAM	0.2 GB

TABLE VI: Consumption per Hour

Resource Leaking	Scenario	Av. (%)	A. Down. (h)
No	Baseline	99.35	56.43
Yes	Opening Transactions - Disk	98.48	133.58
Yes	Opening Transactions - RAM	98.17	160.30
Yes	Query Transactions - Disk	98.88	98.12
Yes	Query Transactions - RAM	98.99	88.47
Yes	Transfer Transactions - Disk	98.57	125.26
Yes	Transfer Transactions - RAM	98.41	139.28

TABLE VII: General Availability Results

At first, **we have evaluated the system's availability without considering the impact of aging issues** on the environment. Then we considered both the disk and RAM consumption as permanent issues and the associated transaction types. The system's general availability was 99.35% with an annual downtime of 56.43 hours, more than two days.

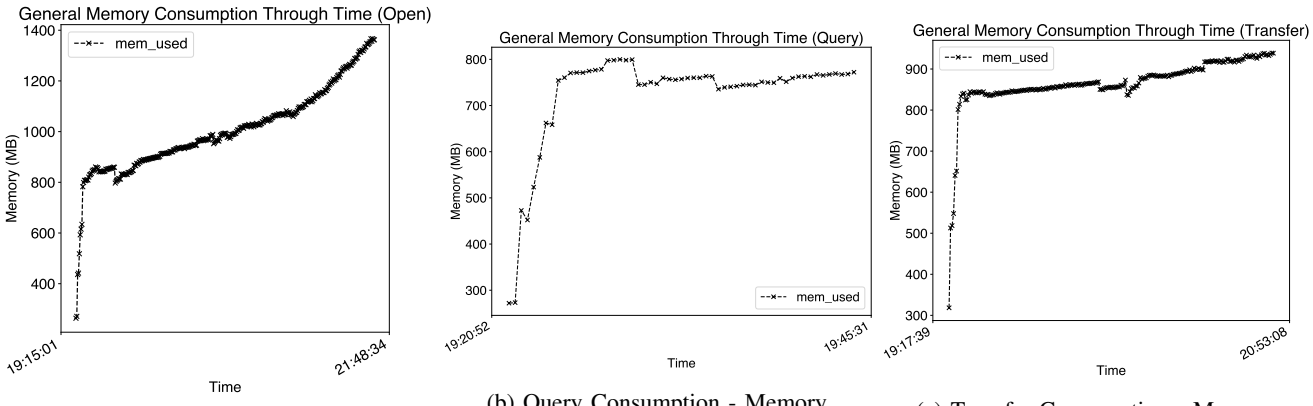
When considering the possibility of resource exhaustion, the evaluated model provides a lower availability, which was already expected. **The opening account transactions** provide the general lowest availability (98.17%) since it is the kind of transaction that most **impacts RAM consumption**.

The query transactions provide the lowest impact on the system's availability among the evaluated transactions, which comes as a surprise. Already transfer transactions are right behind the opening account transaction regarding issues in disk and RAM.

VII. Conclusions and Future Works

This paper evaluated the performance of a Hyperledger Fabric platform deployed over a private environment managed by a single entity. By evaluating the latency (s) and throughput (TPS) of a simple benchmark through the Hyperledger Caliper benchmark tool, we could detect the increase in resource consumption and point out what could be a software aging-related issue.

We considered opening, querying, and transferring between accounts as the main transactions types in the performed benchmark. We measured CPU, disk, cached, and RAM Memory for each kind of transaction. Then we modeled the availability of the system considering an increase in resource consumption, which provided us the impact of the resource consumption over the general system's availability.



(a) Open Consumption - Memory

(b) Query Consumption - Memory

(c) Transfer Consumption - Memory

Fig. 16: RAM Consumed per Transaction Type

As the main limitations of the current work, we may cite that the evaluated scenarios are too specific and do not consider combinations of transactions, such as opening/transfer, opening/transfer/query, opening/query, etc. Another important limitation deals with the performability model that considers the resource exhaustion of a single resource per time, while it is expected the increase in both disk and RAM consumption happened at the same time. Also, we could not pinpoint which component is responsible for the resource leaking, which we plan to perform later.

As future work, we intend to evaluate these limitations and elaborate a software rejuvenation approach that may improve both platform and environment general availability.

ACKNOWLEDGMENT

The authors would like to thank the Brazilian Government for the financial support through the Fundação de Amparo a 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] M. Gupta, *Blockchain for DUMMIES*. John Wiley & Sons, Inc., 2017.

[2] 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.

[3] M. Dabbagh, K.-K. R. Choo, A. Beheshti, M. Tahir, and N. S. Safa, "A survey of empirical performance evaluation of permissioned blockchain platforms: Challenges and opportunities," *computers & security*, vol. 100, p. 102078, 2021.

[4] P. Maciel, J. Dantas, C. Melo, P. Pereira, F. Oliveira, J. Araujo, and R. Matos, "A survey on reliability and availability modeling of edge, fog, and cloud computing," *Journal of Reliable Intelligent Environments*, pp. 1–19, 2021.

[5] S. e. a. Pongnumkul, "Performance analysis of private blockchain platforms in varying workloads," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–6.

[6] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018, pp. 264–276.

[7] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric)," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2017, pp. 253–255.

[8] H. Sukhwani, N. Wang, K. S. Trivedi, and A. Rindos, "Performance modeling of hyperledger fabric (permissioned blockchain network)," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–8.

[9] C. Melo, J. Dantas, D. Oliveira, I. Fé, R. Matos, R. Dantas, R. Maciel, and P. Maciel, "Dependability evaluation of a blockchain-as-a-service environment," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 00 909–00 914.

[10] C. Melo, J. Dantas, P. Pereira, and P. Maciel, "Distributed application provisioning over ethereum-based private and permissioned blockchain: availability modeling, capacity, and costs planning," *The Journal of Supercomputing*, pp. 1–27, 2021.

[11] Hyperledger, "An introduction to hyperledger," Tech. Rep., 2018.

[12] ——, "Introduction to hyperledger business blockchain design philosophy and consensus," Tech. Rep., 2018.

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

[14] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide*. New York, NY, USA: Cambridge University Press, 2000.

[15] Hyperledger, "Hyperledger blockchain performance metrics white paper," Tech. Rep., 2018.

[16] J. L. Henning, "Spec cpu2000: Measuring cpu performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[17] D. Cotroneo, R. Matias Jr, and R. Natella, "Fundamentals of software aging," *Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions*, p. 21, 2021.

[18] A. Avritzer, M. Grottke, and D. S. Menasché, "Software aging monitoring and rejuvenation for the assessment of high availability systems," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020, pp. 327–327.

[19] D. Cotroneo, L. De Simone, R. Natella, R. Pietrantuono, and S. Russo, "A configurable software aging detection and rejuvenation agent for android," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019, pp. 239–245.

[20] M. Torquato, J. Araujo, I. Umesh, and P. Maciel, "Sware: a methodology for software aging and rejuvenation experiments," *Journal of Information Systems Engineering and Management*, vol. 3, no. 2, p. 15, 2018.

[21] P. Maciel, K. Trivedi, R. Matias, and D. Kim, "Dependability modeling," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, 2011.

[22] K. S. Trivedi, S. Hunter, S. Garg, and R. Fricks, "Reliability analysis techniques explored through a communication network example," 1996.

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

[24] 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.

[25] J. Araujo, R. Matos, V. Alves, P. Maciel, F. V. d. Souza, R. M. Jr, and K. S. Trivedi, "Software aging in the eucalyptus cloud computing infras-

tructure: characterization and rejuvenation," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, pp. 1–22, 2014.

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

[27] P. Maciel, R. Matos, B. Silva, J. Figueiredo, D. Oliveira, I. F, R. Maciel, and J. Dantas, "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)*, Jan 2017, pp. 50–57.

[28] P. Pereira, J. Araujo, C. Melo, V. Santos, and P. Maciel, "Analytical models for availability evaluation of edge and fog computing nodes," *The Journal of Supercomputing*, pp. 1–29, 2021.

[29] S. Sebastio, R. Ghosh, and T. Mukherjee, "An availability analysis approach for deployment configurations of containers," *IEEE Transactions on Services Computing*, 2018.