



Towards a Decentralized Blockchain-Based Resource Monitoring Solution For Distributed Environments

Rodrigo B. Dos Passos   [Federal University of Rio Grande do Sul | rpassos@inf.ufrgs.br]

Kassiano J. Matteussi  [Federal University of Rio Grande do Sul | kjmatteussi@inf.ufrgs.br]

Julio C. S. Dos Anjos   [Federal University of Ceará | jcsanjos@ufc.br]

Claudio F. R. Geyer  [Federal University of Rio Grande do Sul | geyer@inf.ufrgs.br]

 Federal University of Rio Grande do Sul - UFRGS - Institute of Informatics - PoBox: 15064 - Av. Bento Gonçalves, 9500 - Porto Alegre, RS, 91501-970, Brazil

Federal University of Ceará, Campi Itapaje, Graduate Program in Teleinformatics Engineering (PPGETI/UFC), Center of Technology, Campus of Pici, Fortaleza, CE, 60455-970, Brazil

Received: 01 November 2023 • **Accepted:** 15 February 2024 • **Published:** 07 March 2024

Abstract The increasing number of connected users and devices to Cloud, Fog, and Edge environments encouraged the creation of many applications and services in the most varied areas and domains. Such services are highly distributed on top of heterogeneous infrastructures that require real-time monitoring. The monitoring process may be considered a complex task since it requires experienced users and robust cloud-based solutions to support the most varied needs in such scenarios. The main problem relies on the centralization of the monitoring approaches for cloud-centric solutions that represent a central point of failure in end-to-end communication, compromising the application's security and performance in case of high latency or downtime. In this context, blockchain networks enable exciting features such as decentralization, immutability, and traceability with higher security levels. This work is towards a blockchain-based and decentralized resource monitoring solution for distributed environments. The proposed solution integrates blockchain technology to continuously monitor, store, and safely broadcast Operating System performance counters in a highly decentralized fashion. The results demonstrated that a blockchain-based monitoring tool based on Smart Contract is feasible and that it may serve as an entry point for varied solutions for monitoring, security, scheduling, and so on.

Keywords: Blockchain, Decentralized Monitoring, Distributed Systems, Distributed Monitoring, Smart Contracts, Hyperledger Fabric

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things	PoS	Proof of Stake
CRUD	Create, Read, Update and Delete	PoW	Proof of Work
DC	Data Center	QoS	Quality of Service
DL	Distributed Ledger	SC	Smart Contract
MPPT	Maximum Power Point Tracking	SCADA	Supervisory Control And Data Acquisition
OS	Operating System	TPS	Transaction per Second
		VM	Virtual Machine

1 Introduction

Thanks to their convenience and economy, cloud-based monitoring solutions have become a significant technology trend.

According to an IDC report, up to 2025, there will be 41.6 billion Internet of Things (IoT) devices with the potential to generate 79.4 ZB of data [Framingham, 2019; Farina *et al.*, 2023]. In this context, the Internet has allowed the integration of many heterogeneous devices in the most varied distributed environments such as clouds, Fog, Edge, clusters, and Data Center (DC) [Samaniego and Deters, 2017].

Despite that, maintaining infrastructure availability, data sharing, and resource management requires constant monitoring of system and application metrics that represent the current infrastructure's utilization. The most known monitoring approach takes place on centralizing the monitoring processes in single cloud servers [Chih-Chen Wang *et al.*, 2006; Hauser and Wesner, 2018; Yahaya *et al.*, 2021; Zhang *et al.*, 2021], representing a bulk of technical, technological, and security challenges.

In some cases, when companies and individuals outsource data to the Internet, they no longer have proper management over it. As a result, untrusted cloud servers could try to observe and control data flows [Zou *et al.*, 2023]. Thus, a single point of failure could represent an important issue for the decision-making process for a wide range of services such as job scheduling tasks, load balancing needs, geo-distribution of data, real-time data processing, scalability services, low latency communication, end-to-end solutions, security of in-

formation, among others [De Souza *et al.*, 2020; Dos Anjos *et al.*, 2021a; Sharma *et al.*, 2018; Dos Anjos *et al.*, 2021b; Matteussi *et al.*, 2022; Anjos *et al.*, 2023].

On the contrary, blockchain technology is an alternative that promotes decentralization as its main benefit, improving trust between entities. It eliminates the third party responsible for managing the interests and storing the information in a decentralized way [Golosova and Romanovs, 2018]. Due to this versatility and information security proposal, blockchain has been applied in different areas, such as supply chain, medicine, and government [Moschou *et al.*, 2020; Liang *et al.*, 2021; Gupta *et al.*, 2023].

Thus, blockchain technology has emerged as an excellent alternative for monitoring computer resources in a secure and decentralized way. This work investigates the feasibility of using blockchain-based technology for decentralized resource monitoring in distributed environments. The proposed solution integrates blockchain technology to provide aware monitoring, storage, and safely broadcast operating system performance metrics of heterogeneous environments in a highly decentralized fashion, avoiding querying all monitored devices to retrieve resource monitor data. Thus, using blockchain, the information of the resource monitor is distributed simultaneously to all devices on the network, avoiding time-based windowed operations for prior information diffusion.

Along these lines, the proposed solution will reproduce a real-world scenario by using a blockchain network with different numbers of nodes on top of the Hyperledger Fabric framework in which workloads will be submitted to verify the behavior of the network regarding performance, stability, and scalability. Lastly, the Hyperledger Caliper performance assessment tool will help gather all needed performance information. The main contributions are summarized as follows:

- A novel and decentralized blockchain-based architecture for resource monitoring in distributed systems that ensures privacy and data protection against cyber-attacks in cutting-edge environments.
- A smart contract-based monitoring solution that provides real-time notifications, ongoing surveillance, and detection and response to infrastructure observability needs.
- A real-world study case demonstrating the feasibility of using blockchain technology for real-time monitoring of computing resources.

This work is structured as follows: Section 2 presents the background; Section 3 shows the related work; Section 4 represents the proposal of this work; Section 5 points out all obtained results of this work. Finally, Section 6 concludes the paper by providing directions for future work.

2 Background

This section discusses traditional and modern monitoring solutions for distributed systems and how to adopt blockchain-based technology for decentralized monitoring in distributed systems.

2.1 Monitoring: from monoliths to modern observability with blockchain

Abderrahim *et al.* defines monitoring services as a critical element of any management system, such as systems that try to automate various processes in the infrastructure but that need to supervise independent components [Anagnostopoulos and Kolomvatsos, 2019]. According to Ward and Barker [2014], a traditional monitoring approach involves collecting relevant states, analyzing those states, and then applying the decision-making process resulting from the data analysis. Still, the widely adopted monitoring tools take advantage of simple programs like the UNIX *df*, *uptime*, or *top* tools used to analyze system-level metrics and execute procedures according to the result obtained.

Hauser *et al.* describe the process of monitoring as the process of alerting and resource utilization profiling of infrastructure with additional application-specific metrics. These metrics help to detect resource shortcomings or overspending of resources such as automatic vertical or horizontal scaling, alerting in case of malfunctions of the deployed applications, and visualization of monitoring data for manual analyses [Hauser and Wesner, 2018].

The monitoring service is critical for any management system and modern applications in highly distributed systems such as clouds, Fog, Edge, clusters, and DC environments. Thus, as the management systems try to automate various processes in the infrastructure, an autonomous module should supervise the independent components to provide high Quality of Service (QoS) to support end users and applications efficiently [Anagnostopoulos and Kolomvatsos, 2019].

However, meeting the desired QoS levels depends not only on the provided services. It also relies on users' and applications' requirements that could be updated over time. Thus, performance variability and availability become a significant concern when a user or application runs on top of a distributed environment on-premise host or consumes a critical service.

In such a context, it is also noticeable that the environment and systems continue to grow in size and complexity, and there is a greater need for tools that automate monitoring processes with little or no human interaction [Hauser and Wesner, 2018]. It means monitoring computing resource tools are moving away from traditional monitoring, built for monoliths, to modern observability [Melnik and Safronkova, 2023] which targets modern applications where the organization gains complete visibility from infrastructure and applications.

The blockchain Distributed Ledger (DL) technology emerged as a new paradigm that allows hosts to synchronize copies of data received from geographically distributed sources [Melnik and Safronkova, 2023; Aste *et al.*, 2017]. Also, according to Aste *et al.* [2017], the DL technologies from which the term originates come from decades ago, initially presented by Haber and Stornetta [1991], as meaning for digitally dated documents to protect against tampering, such as Bhutta *et al.* [2021].

DL allows a complex distributed environment to work with the nearest devices in the ledger network that contains

a copy of data access, reducing not only the time and energy costs associated with data access but also the latency in the network transactions and other related network bottlenecks such as the tendency of centralizing the information retrieved in monitoring systems [Melnik and Safronenkova, 2023].

2.2 Smart Contracts Applied to Resource Monitoring

Blockchain originates from the way transaction data are stored. In this case, in blocks. Thus, the previous block's hash is inserted into the next block's header, where they are linked together, forming a chain [GUPTA, 2017].

Generating a new block is integrally related to the previously generated block through the summary of the data and transactions. It consists of the application of *hash* functions, which uses cryptographic algorithms for its generation. The generation of blocks starts from an initial block, called the genesis block, where the first information of the chain is recorded.

The types of blockchains can be classified as i) public [Bhutta *et al.*, 2021], ii) private [Bhutta *et al.*, 2021], or iii) consortium [Cash and Bassiouni, 2018]. In *Public* blockchains, the participants do not need authorization to participate in the network, representing a truly decentralized process. Thus, participants can participate in the consensus process, read and send transactions, and share the record.

On the other hand, *Private* blockchains are organization-oriented, where participants need authorization to join the network [Cash and Bassiouni, 2018]. In such a case, decentralization is maintained by applying well-defined rules that directly influence the network participants who may be allowed to read or write data on the blockchain.

Finally, Consortium ones, similar to private networks but supporting multiple organizations [Ismail *et al.*, 2019], have a consensus process relatively slower than *Private* but faster than *Public* blockchains.

Behind the hype, the blockchain verifies all transactions by applying a mechanism known as consensus. Consensus evaluates all nodes of a distributed blockchain network into an agreement on a single data set. The consensus problem is a central point in distributed systems, which aim to quickly verify data consistency through the voting of some elected participants [Gu *et al.*, 2021].

Currently, according to Deng *et al.* [2022], the consensus algorithm can be classified into three categories: based on attribute value proof of peers, voting mechanism, and Paxos class, following described.

The algorithm based on proof of value, also known as Proof of Work (PoW), is the central part of Bitcoin's algorithm, where its formula is a combination of dates, versions, block size, and other information, and a random nonce, with a value of (*d*) relating to the difficulty of mining. In this way, as the mining difficulty increases, the target value (*d*) and hash to be found will become increasingly difficult to find.

According to Deng *et al.* [2022], as an alternative to the excessive consumption of PoW resources, the Proof of Stake (PoS) algorithm was proposed along the lines of voting-based algorithms. Each participating node has two attributes in this algorithm: currency holding and currency age. In this

way, new transactions are calculated from the weight based on the age of the coin in each of the nodes, and these same nodes conduct the weight-based election process.

The second classification of consensus algorithms is based on a voting mechanism. The best-known algorithm Within this format is PBFT, where a fault-tolerant Byzantine system is used with digital signatures to guarantee reliability [Deng *et al.*, 2022]. Due to the low efficiency resulting from the constant need to exchange data between nodes with high network requirements, this approach is designed for chains part of a consortium.

Finally, according to Deng *et al.* [2022], the last classification of consensus algorithms is the Paxos Class, where we can highlight the i) Paxos and ii) Raft algorithms. Paxos i) was one of the first distributed consensus algorithms developed and paved the way for many modern and less complex algorithms like Raft. Its main function is to organize the processes that must be chosen to be executed in a distributed environment. Raft represents a distributed consensus algorithm based on Paxos. It is easy to understand and used in blockchain networks like Hyperledger Fabric. In Raft, the consensus is reached in the network through the election of a leader, so it is not a Byzantine Fault Tolerant (BFT) algorithm because the other nodes in the network trust the node that is elected through an election where the total number of votes for the new leader must be $N/2 + 1$, where *N* represents the total number of nodes in the network [Deng *et al.*, 2022].

Additionally, it is also possible to use Solo consensus (the focus of this proposal), where a single node is responsible for arbitrarily approving transactions. Usually, this type of consensus is useful for validating configurations straightforwardly proof of concept [Hyperledger, 2023].

Thanks to integrating consensus algorithms with Smart Contract (SC)s. It is possible to automate the execution and validation of processes. It has become possible to apply such technologies to the monitoring of computer resources, helping to gain reliability in the processing and storage of linked data due to the distributed nature of the processes.

3 Related Work

This section investigates how blockchain has been used to monitor distributed environments such as cloud, fog, and edge environments. In addition, we focus on studying the main characteristics used in such solutions and how they are applied in complex monitoring scenarios.

The work Yang *et al.* [2019] studies the electricity consumption management. The problem behind this study is the amount of energy provided for each consumer versus its usage. The variation may represent an energy consumption gap that varies from 20% to 30%. This gap represents a significant loss for energy service providers regarding power generation potential and distribution.

The authors proposed a blockchain-based system to collect power consumption data using residential smart devices. The devices used blockchain technology to store and share data safely and securely between users and energy service providers in real-time. As a result, the energy service

Table 1. Related Work Comparison

Related Works	Architecture	Monitoring Model	Blockchain Type	Consensus	Fault Tolerance	Smart Contract	Prototype
Yang <i>et al.</i> [2019]	Smart Grid and Blockchain Consortium	Electrical Consumption	Hyperledger Fabric	Not Defined	Gossip Protocol	Energy Level (High/Low). Temperature Level	Yes
Alcarria <i>et al.</i> [2018]	Smart Grid and Blockchain Consortium	Electrical and Water Consumption	Ethereum	PoA	Ghost Protocol	When Changes Occur	Yes
Helebrandt <i>et al.</i> [2018]	Smart Grid and Blockchain Consortium	Configuration Script of Network Devices	Hyperledger Composer and Hyperledger Fabric	Not Defined	Gossip Protocol	No Automatic Interaction	Yes
Košt'ál <i>et al.</i> [2019]	Smart Grid, Blockchain Consortium and Smart Contracts	Devices Configuration History	Hyperledger Composer and Hyperledger Fabric	Not Defined	Gossip Protocol	When Changes Occur	Yes
Xu <i>et al.</i> [2017]	Blockchain Consortium Composed by Data Center	Transactions With Migration Requests	Not Defined	PoW or PoS	Not Defined	With Submission Requests or Allocation Resources.	No
Proposal	Blockchain Consortium Composed by Data Centers Monitoring Virtual Machines	Resource Monitoring	Hyperledger Composer and Hyperledger Fabric	Solo	Gossip Protocol	Automatic	Yes

providers increased efficiency by delivering on-demand energy and providing dynamic consumption reports for their customers.

The proposal of Alcarria *et al.* [2018] studied the different power and water consumption levels between local residential communities. The authors observed a variation in the rates and supply demand for each household. These oscillations incur losses of resources and represent a lack of optimization and management of resources among neighbors. It is because a low consumption quota of a client will not be directly used by another client, which could be considered a benefit for the community.

The authors proposed a blockchain system that individually measures and manages resources for each residence in real-time. The proposed system worked based on users' statistics collected by smart devices, corresponding to their consumption patterns. Thus, the residents may negotiate with the energy service providers and manage resource credits. It means the surplus energy can be sold to another user who exceeded his quota.

Usually, specific network device configurations are saved in a database and uploaded to the device after any change. This approach requires a monitoring strategy, which can be active or passive. In the active format, metrics are collected, and this causes additional network traffic or increased demand for computing resources. The passive format has no additional traffic but requires monitoring any metrics not updated on the device.

Thus, the authors Helebrandt *et al.* [2018] suggested the use of blockchain to perform the monitoring in an automated fashion, where the administrators submit changes in the configuration files to specific devices, and the devices themselves verify the existence of any changes in periods of 2 minutes. If a device detects a change, it retrieves its configuration script from the blockchain and updates itself automatically.

The work proposed by Košt'ál *et al.* [2019] studied the high growth of IoT devices and sensors. The unexpected growth increased the need for network security during data transmission, efficient management, and support. However, security and management must work alongside each other to support issues like unauthorized access or unexpected issues that spawn after inserting too many devices into a new system.

The proposed system used blockchain to provide storage

for the device configurations. This process used SCs to manage Create, Read, Update and Delete (CRUD) operations in the database. Thus, by using blockchain in such a way, the solution allowed a quick recovery for the environment in case of system incidents.

The work Xu *et al.* [2017] focuses on the migration of requests from traditional data centers to data centers with renewable energy, aiming to minimize environmental impact. According to the authors, it represents a trend adopted by large corporations such as Google, Amazon, and Microsoft that aim to use renewable energy sources to host their data centers and reduce energy costs. Thus, taking as a reference the high energy consumption of the data centers of these companies. The authors proposed a framework based on blockchain technology that works as the coordinator, which uses reinforcement learning as an entry point system to migrate these requests to green data centers.

3.1 Discussion

Table 1 presents the main characteristics of each related work such as architecture, the monitoring model used, blockchain type, consensus algorithm and fault tolerance applied as well as how the SC are used, the existence of a prototype and, finally, a comparison with the current work.

The state-of-the-art demonstrated blockchain technology to decentralize information, allowing data availability in a geo-distributed manner. In such a context, this work takes advantage of the decentralized capabilities of blockchain to overcome the standard local and centralized infrastructure monitoring solutions by providing a decentralized resource monitoring tool for distributed systems.

The proposed work differs from others since it allows direct and dynamic interaction with the blockchain according to the resource monitoring needs of a distributed environment, like providing resource information related to a private cloud comprised within several DC that receive VM requests for many Virtual Machines Virtual Machine (VM)s and need to perform scheduling actions in real-time.

Finally, the applicability of the SCs varies too. The related works do not allow direct iteration within the client according to specific requests and direct submission to the blockchain. Conversely, the proposed solution decentralizes data storage and monitoring, allowing any node participating in the blockchain network to check the status of the comput-

ing resources of any other VM at any time.

4 Model

This work aims to provide a solution capable of monitoring distributed environments in a decentralized fashion. As a motivational case study, we can figure out a distributed cluster used by a private cloud. In such a scenario, clients may launch several VMs in parallel, and the cluster needs to dispatch the request by analyzing multiple features such as resource availability and utilization, Operating System (OS) load, network performance, storage, and so on. Still, all the communication between DCs must happen even when a node or device is left behind due to an unintended issue. In the following, this section presents all the architectural elements needed to provide a blockchain-based resource monitoring solution.

4.1 Architecture and Design

It is possible to observe in Figure 1 a blockchain-based architecture interconnecting distributed *Peers* (DCs) in a decentralized fashion. The data monitoring is designed using SCs for safe data persistence and commitment on the blockchain. The monitoring traces represent a valuable input for the cloud scheduler since it must dispatch user requests as quickly as possible. Furthermore, it is essential to highlight that the blockchain provides fast data recovery, availability, and security features, always keeping a faithful copy of the data on all the nodes participating, and a failure in any of these nodes does not compromise the rest of the network [Salama *et al.*, 2023].

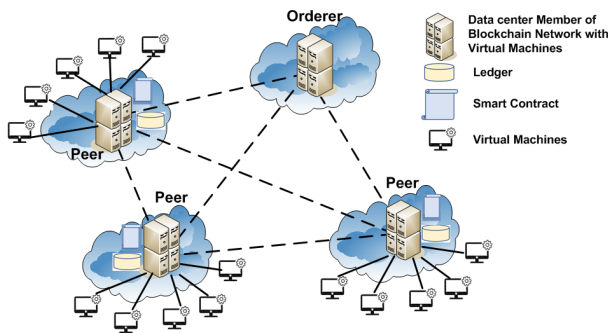


Figure 1. Decentralized Monitoring Architecture Applied to a Data Center Scenario

Each *Peer* will receive requests from its internal network comprising all the monitored VMs. The distribution of the data that was generated among all DCs is performed by the *Orderer* node, in a way that each *Peer* that receives the new block sent by the *Orderer* node must validate it and update its local blockchain copy, as shown in Figure 2. Still, it is important to mention the *Peer* nodes will also participate in the consensus task for transactions performed in other DC.

According to Figure 2, the DCs are represented by *NodeN* that hosts several VMs from a DC. Each VM is monitored independently and continuously communicates with a *Peer* container that belongs to the Hyperledger Fabric's structure.

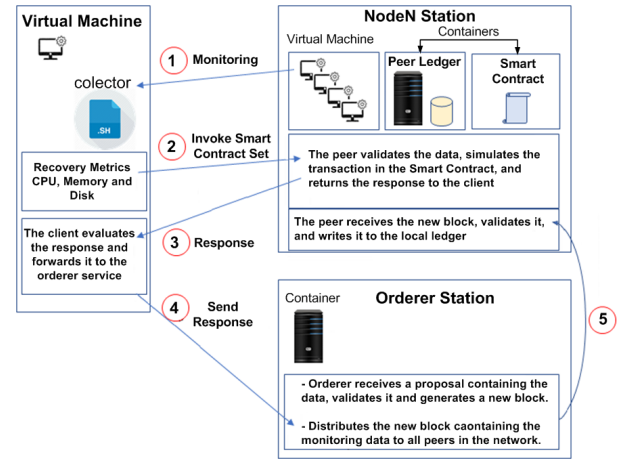


Figure 2. Model Overview

Each monitored VM executes once per second a script file called *collector.sh* that retrieves CPU, memory, and disk metrics. After the resource information retrieval, the script performs an *Invoke* call to *Peer* container, starts the validation process, and interacts with the correspondent SC to execute the transaction.

Following this, the output is returned to the client for a reevaluation. If this answer is correct, it is forwarded to the *Orderer* for evaluation, creating a new block for distribution. After the creation of the new block by the *Orderer*, this block is distributed to all *Peers* of the network. Then, each *Peer* is responsible for updating the blockchain locally with the new information regarding the retrieved resources.

4.2 Smart Contract Applied to Resource Monitoring

The SC represents the technology enabling blockchain interaction. Each SC is installed in a channel and interconnects all peers, allowing resource monitoring in real-time. A SC receives requests originating from properly authorized clients and records them into the blockchain. This work proposed a bulk of applications to interact with the blockchain, providing a way to monitor and verify computer resources through the use of SC functions. Table 2 describes the SC functions, type, and their objective in this work.

According to Table 2, this work provides two kinds of applications. The *writing applications* that use a function to append monitored data to the blockchain, e.g., *set* application. The *Reading applications* that uses a function that does not alter the blockchain and performs only reading operations to retrieve information data from raw monitored resources in the blockchain, e.g., *getTotalMonitoredData*, *getEquipmentMonitoredData*, *getEquipmentData*, *getEquipmentLastStates* application.

Depending on the read or write application, the framework uses different steps in the consensus process (Endorsement). Independently of the SC being used, the ordering format applied to the consensus process is the type *Solo*, which contains a single *Orderer* in the network responsible for validation, creation, and distribution of new blocks. Even though this consensus format is recommended only for development environments, it is also recommended for perfor-

Table 2. Implemented Smart Contracts

Smart Contract	Function	Purpose	Complexity
set	Add resource monitoring data in the blockchain.	Write	$O(1)$
getTotalMonitoredData	Retrieve data about CPU, memory, and disk of all network nodes.	Read	$O(n)$
getEquipmentMonitoredData	Obtains the CPU, memory, and disk data from a specific network node.	Read	$O(1)$
getEquipmentData	Returns all nodes that match a defined input.	Read	$O(n)$
getEquipmentLastStates	Returns the last records of certain equipment, up to 10 records.	Read	$O(n)$ $\Theta(n \log(n))$

mance tests to verify the viability of monitoring resources by using blockchain technology as a basis for it. It was made because scenarios with one *Orderer* result in the worst possible case and the slightest fault tolerant. Therefore, adding more *Orderer* reduces the probability of bottlenecks in the *Orderer*, thus increasing the network's performance and fault tolerance.

4.2.1 Writing Applications

Figure 3 presents the whole workflow for the consensus process of a writing application. As in the reading transactions, the writing process begins with the endorsement stage. It validates this endorsed proposal to verify if it is incorruptible compared to the initial request, represented in blue. Then, after the proposal's endorsement, the client forwards the proposal for validation and creation of the block for storing this transaction, represented by the yellow arrows. After creating a new block, it must be spread to all the network participants, updating the blockchain, represented by the green arrows. The last step is communication with the client, who will receive the responses to the requests specifying that the transaction was correctly submitted, represented by the gray arrow.

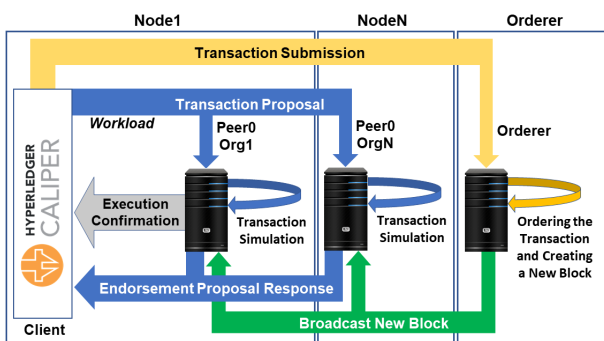


Figure 3. The Endorsement of writing applications is represented by different colors: the Endorsement process is represented in blue, the Ordering process in yellow, and the Validation/Submission process in green.

As seen on Algorithm 1, it performs the validation if the function receives the list of arguments with two items: the first is the name of the VM, and the second represents the values of associated monitoring. Then, the function generates the updated data and substitutes the wildcards that exist on the value field that were inserted by the resource monitor located at each VM using the current date and time. In the next step, the updated data are recorded in the blockchain using

the specific Hyperledger Fabric *PutState* function, followed by an error verification. Finally, the confirmation message is returned to the requester.

Smart contract function **Set** is responsible for writing monitoring data in the blockchain, as presented by Algorithm 1.

Algorithm 1 Set Function

```

1: procedure set(args)
2:   if QUANTITY(args) ≠ 2 then
3:     return "Arguments number incorrect."
4:   end if
5:   Format current Date
6:   Put date on wildcards
7:   PutState function to Record data on the chain
8:   if CHECK_ERROR = True then
9:     return "Error writing on the blockchain."
10:  end if
11:  Return Write Confirmation
12: end procedure

```

4.2.2 Reading Applications

While executing the reading functions that do not alter the Ledger, the Orderer does not participate in the validation process. Figure 4 represents the whole workflow for the consensus process of a reading application.

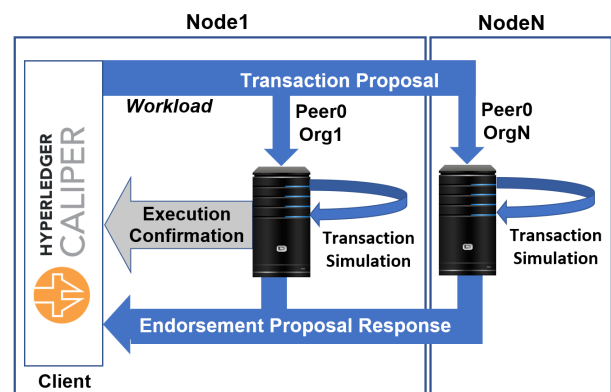


Figure 4. Read Operation Process

Figure 4 represents the execution process of reading transactions that contain only the Endorsement phase. In this process, the client represented by the Caliper Hyperledger triggers the initial transaction flow, which will assemble a transaction proposal over one of the available functions for the requisition workload. Then, the client submits this proposal

for the endorsement policy for every peer-related ot. Since it is a reading operation, the endorsed data is validated to check if it is incorruptible compared to the initial request. Thus, if properly validated, it is returned to the client for use. The implemented SC is presented below:

Smart contract function **getTotalMonitoredData** is responsible for returning the monitoring data of all blockchain network participants. It is represented by Algorithm 2.

The Algorithm 2 starts performing the recovery of the monitoring values of all existing VMs in the blockchain, using a specific data recovery function belonging to the Hyperledger Fabric called *GetStateByRange*. Since this function returns the data in object format, converting it to JSON format is necessary, which is done inside the loop by concatenating strings. Finally, the JSON is returned with monitoring data of all VMs of the blockchain.

Algorithm 2 getTotalMonitoredData Function

```

1: procedure getTotalMonitoredData
2:   totalKeys ← GetStateByRange()
3:   if CHECK_ERROR = True then
4:     return "Operation to search keys failed."
5:   end if
6:   for (totalKeys has more Keys = True) do
7:     object ← GetStateByRange
8:     if CHECK_ERROR = True then
9:       return "Error retrieving keys."
10:    end if
11:    jsonReturn ← jsonReturn + "{ +
object.key + : + object.value + }"
12:  end for
13:  Return jsonReturn
14: end procedure

```

Smart contract function **getEquipmentMonitoredData** is responsible for returning the resource monitoring data of a specific VM. The Algorithm 3 represents the function.

Algorithm 3 getEquipmentMonitoredData Function

```

1: procedure getEquipmentMonitoredData(args)
2:   if QUANTITY(args) ≠ 1 then
3:     return "Arguments number incorrect."
4:   end if
5:   value ← GetState(args)
6:   if value = NULL then
7:     return "Equipment search failed."
8:   end if
9:   Return value
10: end procedure

```

According to the Algorithm 3, it starts checking if the VM name is passed as the only function parameter. Otherwise, it returns an error to the requester. Then, the algorithm performs a search in the blockchain using the *GetState* function, which retrieves a specific key in the chain. If no error occurs during the retrieval process, the monitoring data of the equipment that was informed is returned.

Smart contract function **getEquipmentData** returns the VMs according to the previous monitoring metrics as arguments. The function's representation is shown in Algorithm 4.

Algorithm 4 getEquipmentData Function

```

1: procedure getEquipmentData(args)
2:   jsonReturn ← jsonReturn + "["
3:   totalKeys ← GetStateByRange()
4:   if CHECK_ERROR = True then
5:     return "Error retrieving keys."
6:   end if
7:   for (totalKeys hasMoreKeys = True) do
8:     object ← totalKeys.key
9:     if object = NULL then
10:      return "Error retrieving key."
11:    end if
12:    Convert object to structure
13:    if args[0] = "CPU" OR args[0] = "MEM" then
14:      Retrieve first value from struct
15:      Retrieve second value from struct
16:      Search equip by first and second value
17:      if jsonReturn ≠ "[" then
18:        jsonReturn ← jsonReturn + ", "
19:      else
20:        jsonReturn ← jsonReturn +
object.Key
21:      end if
22:      else if args[0] = "STG" then
23:        Retrieve first value from struct
24:        Retrieve second value from struct
25:        Searching equipment according to first and second
value
26:        if jsonReturn ≠ "[" then
27:          jsonReturn ← jsonReturn + ", "
28:        else
29:          jsonReturn ← jsonReturn +
object.Key
30:        end if
31:      end if
32:    end for
33:    jsonReturn ← jsonReturn + "]"
34:    Return jsonReturn
35: end procedure

```

The Algorithm 4 starts retrieving all monitoring data of all VMs stored in the blockchain through the call *GetStateByRange* of the Hyperledger Fabric. After checking for errors during the call, the algorithm performs iteration of all stored VMs, converting their monitoring data and comparing them with the monitoring data that were informed in the function call. If some equipment of the list meets the requirements of metrics values between the initial and final values informed in the search, then this VM is added to the list for return upon the request's end. By the end of the loop, the return will contain all the equipment checked out on the input's specific criteria.

Smart contract function **getEquipmentLastStates** is responsible for the recovery of the last 10 states of a specific VM, as portrayed in Algorithm 5.

Then, the Algorithm 5 proceeds to obtain the history of the informed VM by using the Hyperledger Fabric's *GetHistoryForKey* function and verifying the success of the call. Then, the algorithm creates a structure list for storing the retrieved values, which are iterated and stored individually in the list. Once the list is filed entirely, the algorithm executes a sorting by date of inclusion in the blockchain. Finally, this sorted list is iterated ten times, storing each element into a

Algorithm 5 getEquipmentLastStates Function

```

1: procedure getEquipmentLastStates(args)
2:   if QUANTITY(args)  $\neq$  1 then
3:     return "Arguments number incorrect."
4:   end if
5:   history  $\leftarrow$  GetHistoryForKey()
6:   if history = NULL then
7:     return "History search failed."
8:   end if
9:   Create list of structure
10:  historyList  $\leftarrow$  StructureNode()
11:  for (history hasMoreKeys = True) do
12:    response  $\leftarrow$  history.key()
13:    historyList  $\leftarrow$  historyList + response
14:  end for
15:  SORT(historyList)
16:  for (history hasMoreKeys = True) do
17:    jsonStr  $\leftarrow$  history.key()
18:    if counter  $\geq$  LastRequests() then
19:      BREAK
20:    end if
21:    counter  $\leftarrow$  counter + 1
22:  end for
23:  Return jsonStr
24: end procedure

```

JSON string to be returned to the requester.

4.2.3 Complexity Analysis

A complexity analysis from the proposed algorithms is summarized as shown in Table 2. Follows a brief discussion about them.

- Algorithm 1 - SC **set** demonstrates a complexity of $O(1)$. Each transaction represents a write operation with well-defined input parameters for monitoring data.
- Algorithm 2 - SC **getTotalMonitoredData** demonstrates a complexity of $O(n)$ due to the need to load all the metrics stored for each piece of equipment on the blockchain and then iterate them to be returned after preparation.
- Algorithm 3 - SC **getEquipmentMonitoredData** demonstrates a complexity of $O(1)$. This algorithm performs a specific search using the equipment data entered as input, so the complexity does not vary, remaining at $O(1)$.
- Algorithm 4 - SC **getEquipmentData** demonstrates a complexity of $O(n)$. The costs are associated with the iterations over the entire blockchain to search for equipment whose metrics meet the conditions applied as input. Thus, the time cost increases as the data stored on the blockchain increases proportionally.
- Finally, Algorithm 5 - SC **getEquipmentLastStates** demonstrates a complexity of $O(n)$ in the best case when the *history lists* are already ordered and $(n \log(n))$ in the worst case. The mean case costs are obtained through the iterations required to traverse the blockchain and search for the last states. The worst case comes from the need for the ordering results to be presented chronologically.

4.3 Summary and Discussion

The proposed monitoring tool is not only restricted by the *set* write function used for the monitoring process but also by reading functions that enable retrieval of information stored in the blockchain. Thus, the model created allows the development of applications with different filters to obtain specific details on the status of each VM regarding the individual utilization of resources of these machines and identify a set of VMs that meet specific criteria.

The proposed model uses a tool called *dstat* for monitoring OS metrics. However, it is important to mention that the proposed solution was conceived to be generic, and other tools may be used, obeying the necessary adjustments to integrate with the SC.

5 Evaluation

The evaluation section investigates the viability of using blockchain technology as a monitoring tool for distributed environments. This section presents the hardware and software stack used to perform the experiments and the methodology chosen to evaluate and analyze the results of this work.

5.1 Methodology

The methodology of this work aims to demonstrate a real-world monitoring scenario using distributed DCs comprised of many nodes and varied loads, e.g., VMs for monitoring. In this context, to carry out the experiments, four scenarios were proposed with different amounts of DCs and VMs that will transact with each other. Regardless of the scenario presented, the execution of the experiments is triggered by the Hyperledger Caliper from the first DC on the network (*node 1*). The *node 1* was created with the greater computational capacity to produce, dispatch, and record a massive number of requests. The other DCs on the network have similar hardware configurations as they do not run Hyperledger Caliper instances for load generation.

The first scenario comprises three transactional DCs and an ordering node. This scenario allows the execution of a monitoring request per running VM, increasing the number of VMs per DC and transactions per second linearly. The other test scenarios follow the same logic regarding the increment of VMs and transactions per second, differing only in the number of DCs. The consensus chosen to validate all the transactions in the network is known as Solo, in which only the Orderer verifies the transactions from the whole network. In this way, we can observe the evolution of the configuration of the experiments according to Table 3.

Caliper allows generating reports containing some performance indicators [Hyperledger, 2022a]. Currently, indicators supported by this framework are latency for the transaction/read, throughput for the transaction/read, and consumption of resources such as CPU, memory, and network [Hyperledger, 2022b].

Table 3. Performance Evaluation Scenarios Overview

	#DC	Orderers	VMs per data center	Monitored (VMs)
Scenario 1	3	1	1,2,3,4,5,6,7,8,9,10,20	3,6,9,12,15,18,21,24,27,30,60
Scenario 2	7	1	1,2,3,4,5,6,7,8,9,10,20	7,14,21,28,35,42,49,56,63,70,140
Scenario 3	11	1	1,2,3,4,5,6,7,8,9,10,20	11,22,33,44,55,66,77,88,99,110,220
Scenario 4	19	1	1,2,3,4,5,6,7,8	19,38,57,76,95,114,133

5.2 Software and Hardware

All nodes were installed on top of Ubuntu 18 LTS operating system, Docker container 20.10.12, version Hyperledger Fabric 2.3.0, node.js 10.x, Docker Compose 1.21.2, Golang language 1.15.7, and to evaluate the blockchain performance, Hyperledger Caliper 0.4.2 was used. The environment set up to run the prototype is part of a cluster maintained by UFRGS¹, using the OpenStack tool.

To carry out the experiments, 20 VMs were created to represent physical nodes in this work, where one VM is responsible for performing the benchmark tests and being part of the blockchain. The other 19 are part of the DC that will transact on the blockchain network. The VM called node1 is responsible for running the blockchain network benchmark tests focused on monitoring resources and is comprised of 8 vCPUs, 16GB RAM, and 300GB of disk space. The other 19 VMs in the blockchain network are comprised of 4 vCPUs, 4GB RAM, and 100GB of disk space.

5.3 Results

Table 4 presents the outcomes of two sets of tests. Initially, the results related to 1VM per DC demonstrated the stability of sending requests on all SC operations for scenarios 1, 2, and 3. In such case, a constant sending rate of approximately one Transaction per Second (TPS) per VM with a slight disparity between them throughout each scenario. Conversely, scenario 4 presented a degradation up 20% in latency and throughput due to the lack of *orderers* to validate transactions. Based on our estimations, one additional *orderer* may be enough to steady performance and alleviate the obtained degradation.

Still on Table 4, but looking at the results comprised by 7 VMs per DCs. It is possible to observe the latency increased significantly, and sudden drops in throughput values occurred, mainly for scenarios 3 and 4, representing 77 and 133 monitored VMs, respectively. The loss in performance is most related to the consensus process of the network since most of the Peers perform validations.

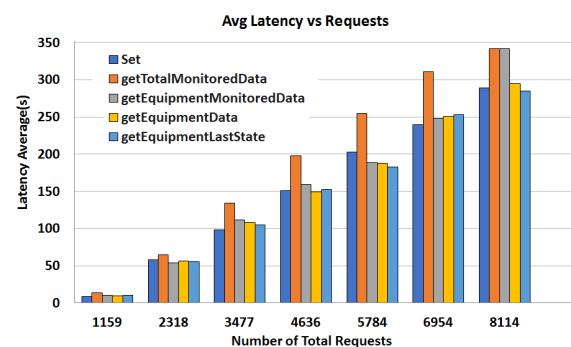
Beyond that, it is important to remark there is a higher latency and throughput for writing operations because there is a higher overhead than reading operations in the environment. This overhead varies around 5.5 times during the oscillation between 1 VM per DC and 7 VMs per DC, caused mainly by the need to check consistency and synchronicity.

In the following, this section describes a more detailed assessment of scenario 4 (Table 4), the most intensive evaluation scenarios # number of Peers with associated VMs. This scenario comprised 19 Peer nodes and one Orderer node. The

VMs varied between 1 and 7 per Peer, totaling 133 monitored VMs.

Figure 5 presents a significant increase in latency due to the growth in the number of Peers in the network and the number of associated VMs. It led to an average latency of more than 300 seconds in experiments using 133 VMs with a maximum value of 8100 requests. In this scenario, a transaction sending rate was maintained up to 120 TPS. It is also possible to observe that the reading and writing operations do not present significant differences. The main variation occurred in the **getTotalMonitoredData** read function that needs more computational effort to retrieve all the monitored data from all the VMs. The **getEquipamentMonitoredData** presents some variation as it retrieves the equipment's monitoring data according to a search of the whole chain.

A variation in latency was observed during the execution of the experiments carried out by Hyperledger Caliper, which is responsible for executing the requests to run the functions provided by the proposed monitoring tool. As demonstrated, high consumption of resources occurred due to the parallelization of the processes for executing Caliper framework requests, increasing even more with a growing number of VMs on the network.

**Figure 5.** Comparison between Average Latency vs Requests

As observed in other test scenarios, the CPU and memory consumption variation in the chain code container was extremely low, which did not differ in this scenario either. Still, in the peer container, it remained very high since the initial tests with 19 Peers and 57 VMs, launching a total of 2014 requests, which had already exceeded 100% CPU utilization on average and peaked with more than 200% utilization, as shown in Figure 6.

¹Federal University of Rio Grande do Sul

Table 4. Comparison between Scenarios

VMs per DC	Operations	Average Latency (s)				Throughput (TPS)			
		Scenarios				Scenarios			
		1	2	3	4	1	2	3	4
1 VM	Set	1.33	0.91	0.80	8.88	3.00	7.00	10.70	15.00
	getEquipmentData	1.29	0.91	0.80	9.39	3.00	6.90	10.80	14.80
	getEquipmentLastState	1.32	0.91	0.82	10.35	2.90	7.00	10.80	15.30
	getEquipmentMonitoredData	1.27	0.91	0.81	10.41	3.00	6.90	10.80	14.90
	getTotalMonitoredData	1.31	0.91	0.81	13.62	3.00	6.90	10.80	13.80
7 VMs	Set	0.47	3.22	66.19	341.92	20.60	46.00	28.60	14.20
	getEquipmentData	0.46	2.25	58.75	284.99	20.50	44.00	28.40	14.00
	getEquipmentLastState	0.46	2.25	58.75	284.99	20.60	46.50	27.60	14.50
	getEquipmentMonitoredData	0.47	3.96	55.91	295.06	20.60	44.40	28.20	12.00
	getTotalMonitoredData	0.46	3.02	58.92	341.92	20.50	45.40	26.20	12.00

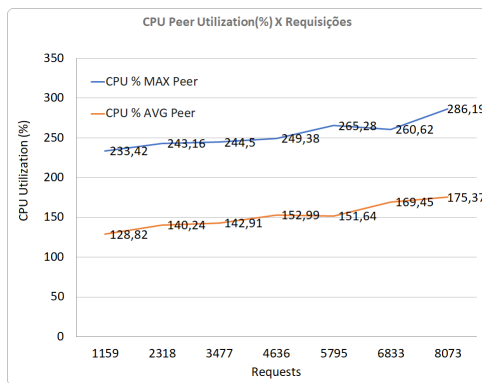


Figure 6. Comparison between CPU Utilization vs Requests

The high CPU utilization occurs because the framework responsible for experimentation is allocated to one of the Peers whose performance is being evaluated. Thus, the set of requests triggered from this framework to this Peer causes a large consumption of resources at the same time as the peer validates transactions and updates them locally, causing an overload in CPU consumption and, as a consequence, a delay in processing the requests.

On the other hand, the memory used by the Peer is directly related to the number of requests, reaching almost 1 gigabyte of memory consumed for a total of 5784 requests. It is possible to observe a sudden drop and rise in consumption due to a small drop in CPU utilization for the number of requests reported, perhaps due to an overload on other peers, allowing fewer local endorsement and recording operations to be carried out, as shown in Figure 7.

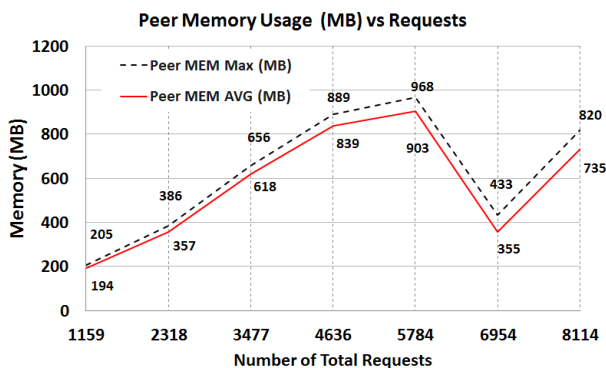


Figure 7. Comparison between Peer Memory vs Requests

Figure 8 presents a slight variation in CPU usage for the chaincode container, up to 8144 requests, maintaining oscillations of no more than 2% use of the computing potential until the end of the experiments.

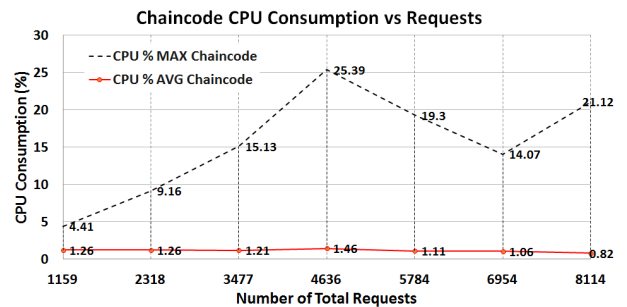


Figure 8. Comparison between Chaincode CPU Consumption vs Requests

This slight variation is due to the manipulated data, which is only a few bytes, and the simplicity of the functions used to manipulate the information, which did not require a significant load within their methods.

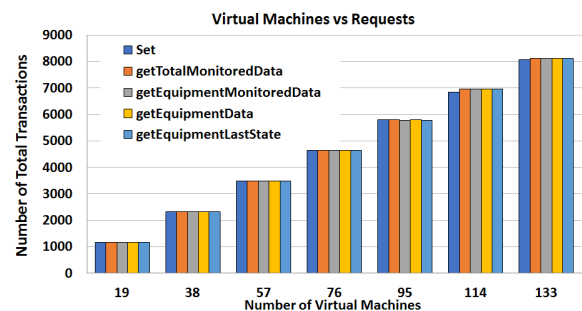


Figure 9. Comparison between Number of Virtual Machines vs Requests

Figure 9 shows the number of requests grows linearly as the number of VMs increases, with practically no variation between read and write operations on the chaincode.

Finally, according to Figure 10, there is a slight variation in Throughput as the number of Peers in the network increases, with a lower Throughput for the functions responsible for handling a large amount of data to obtain feedback.

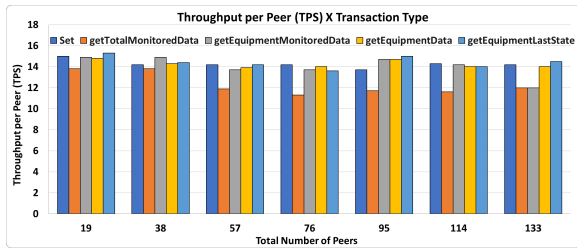


Figure 10. Comparison between Throughput per Peer vs Transaction type

6 Conclusion and Future Work

This work investigated the feasibility of using blockchain-based technology for decentralized resource monitoring in distributed environments. It was possible to observe that the proposed solution successfully integrated blockchain technology and SC to provide aware monitoring, even for highly intensive workloads. As a result, multiple copies of the data are spread over the network, facilitating information access.

It is important to highlight the outcomes revealed blockchain operated stable and at scale. Although some latency and throughput degradation were detected, it does not mean the proposed monitoring model is non-performable. Conversely, in order to retrieve the maximum performance and avoid high latency and throughput, it is necessary to provide more *orderers* in the network proportionally to the network size by following $(TotalNodes/2) + 1$ [Hyperledger, 2023].

Still, it was observed the performance of Caliper framework. Even though it represents a powerful tool for measuring blockchain performance, the use of resources was high, leading to some interference in the obtained results. Thus, the monitoring process becomes costly when associating Caliper with a larger number of analyzed data centers. Thus, changing or configuring the Caliper tool to work in a distributed manner and including more *orderers* in the network could improve performance.

Finally, we believe this work may contribute to the research community in varied ways, such as providing a solution that serves as the basis not only for the monitoring process but also to secure information like cloud accountability, to serve as an entry for scheduling algorithms, to enforce security in transactions and so on.

Although the initial proposal to use a blockchain platform to monitor computing resources has been met, further studies can be applied to improve performance in general and, consequently, the results obtained. Therefore, here are some suggestions for future work that could build on the work already done:

- Proceed with the increase in the number of computers in the network, enabling a decentralized validation process, thus verifying the impact on the latency and throughput indicators presented.
- Proceed with decentralizing the performance evaluation software Hyperledger Caliper, allocating it to different network nodes to provide greater competition in creating tasks to perform tests on the created SC.
- Experiment with different controllers for request submission rates provided by the Hyperledger Caliper tool,

to test the limits of the network under other circumstances.

- Experiment with different consensus algorithms for the computer, to verify the impact on the throughput of transactions, comparing with the results of the Solo controller tested for this proposal.
- Verify the performance variation and interference by comparing with traditional monitoring tools such as Fluentd.

Acknowledgements

National funds sponsored this work through CEI UFRGS, SmartSent (#17/2551-0001 195-3), CAPES (Finance Code 001), CNPq, PROPESQ-UFRGS-Brazil, program PRONEX 122014. FAPERGS Project GREEN-CLOUD - Computação em Cloud com Computação Sustentável (#16/2551-0000 488-9). PNPd program grant #2021/11325-4, São Paulo Research Foundation (FAPESP), CERIEA Project (#2020/09706-7) FAPESP–MCTIC-CGLBR in partnership with Hapvida NotreDame Intermédica group.

Declarations

Authors' Contributions

Conceptualization, dos Passos, R.B.; Matteussi, K.J. and Geyer, C.F.R.; methodology dos Passos, R.B.; Matteussi, K.J.; dos Anjos, J.C.S. and Geyer, C.F.R.; software dos Passos, R.B. and Matteussi, K.J.; validation dos Passos, R.B.; Matteussi, K.J. and dos Anjos, J.C.S.; writing—original draft preparation dos Passos, R.B.; Matteussi, K.J.; dos Anjos, J.C.S. and Geyer, C.F.R.; writing—review and editing, dos Passos, R.B.; Matteussi, K.J.; dos Anjos, J.C.S. and Geyer, C.F.R.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

Data can be made available upon request.

References

- Alcarria, R., Bordel, B., Robles, T., Martín, D., and Manso-Callejo, M.-Á. (2018). A blockchain-based authorization system for trustworthy resource monitoring and trading in smart communities. *Sensors*, 18(10):3561. DOI: 10.3390/s18103561.
- Anagnostopoulos, C. and Kolomvatsos, K. (2019). An intelligent, time-optimized monitoring scheme for edge nodes. *Journal of Network and Computer Applications*, 148:102458. DOI: 10.1016/j.jnca.2019.102458.
- Anjos, J. C. S. d., Matteussi, K. J., Orlandi, F. C., Barbosa, J. L. V., Silva, J. S., Bittencourt, L. F., and Geyer, C. F. R. (2023). A survey on collaborative learning for intelligent

- autonomous systems. *ACM Comput. Surv.* Just Accepted. DOI: 10.1145/3625544.
- Aste, T., Tasca, P., and Di Matteo, T. (2017). Blockchain technologies: The foreseeable impact on society and industry. *Computer*, 50(9):18–28. Available at: https://discovery.ucl.ac.uk/id/eprint/10043048/1/Aste_BlockchainIEEE_600W_v3.3_A.docxceptedVersion.x.pdf.
- Bhutta, M. N. M., Khwaja, A. A., Nadeem, A., Ahmad, H. F., Khan, M. K., Hanif, M. A., Song, H., Alshamari, M., and Cao, Y. (2021). A survey on blockchain technology: Evolution, architecture and security. *IEEE Access*, 9:61048–61073. DOI: 10.1109/ACCESS.2021.3072849.
- Cash, M. and Bassiouni, M. (2018). Two-tier permissioned and permission-less blockchain for secure data sharing. In *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 138–144. DOI: 10.1109/SmartCloud.2018.00031.
- Chih-Chen Wang, Yung-Mu Chen, Cheng-Hao Weng, and Tein-Yaw Chung (2006). An overlay resource monitor system. In *2006 8th International Conference Advanced Communication Technology*, volume 3, pages 5 pp.–1879. DOI: 10.1109/ICACT.2006.206358.
- De Souza, P. R. R., Matteussi, K. J., Veith, A. D. S., Zanchetta, B. F., Leithardt, V. R. Q., Murciego, A. L., De Freitas, E. P., Anjos, J. C. S. D., and Geyer, C. F. R. (2020). Boosting big data streaming applications in clouds with burstflow. *IEEE Access*, 8:219124–219136. DOI: 10.1109/ACCESS.2020.3042739.
- Deng, X., Li, K., Wang, Z., Li, J., and Luo, Z. (2022). A survey of blockchain consensus algorithms. In *2022 International Conference on Blockchain Technology and Information Security (ICBTIS)*, pages 188–192. DOI: 10.1109/ICBTIS55569.2022.00050.
- Dos Anjos, J. C., Gross, J. L., Matteussi, K. J., González, G. V., Leithardt, V. R., and Geyer, C. F. (2021a). An algorithm to minimize energy consumption and elapsed time for iot workloads in a hybrid architecture. *Sensors*, 21(9):2914. DOI: 10.3390/s21092914.
- Dos Anjos, J. C. S., Gross, J. L. G., Matteussi, K. J., González, G. V., Leithardt, V. R. Q., and Geyer, C. F. R. (2021b). An algorithm to minimize energy consumption and elapsed time for iot workloads in a hybrid architecture. *Sensors*, 21(9). DOI: 10.3390/s21092914.
- Farina, M. D., dos Anjos, J. C., and de Freitas, E. P. (2023). Real-time auto calibration for heterogeneous wireless sensor networks. *Journal of Internet Services and Applications*, 14(1):1–9. DOI: 10.5753/jisa.2023.2739.
- Framingham, M. (2019). The growth in connected iot devices is expected to generate 79.4zb of data in 2025. Available at: <https://www.businesswire.com/news/home/20190618005012/en/The-Growth-in-Connected-IoT-Devices-is-Expected-to-Generate-79.4ZB-of-Data-in-2025-According-to-a-New-IDC-Forecast>.
- Golosova, J. and Romanovs, A. (2018). The advantages and disadvantages of the blockchain technology. In *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, pages 1–6. DOI: 10.1109/AIEEE.2018.8592253.
- Gu, W., Li, J., and Tang, Z. (2021). A survey on consensus mechanisms for blockchain technology. In *2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA)*, pages 46–49. DOI: 10.1109/CAIBDA53561.2021.00017.
- GUPTA, M. (2017). *Blockchain IBM Limited Edition*. John Wiley Sons, Inc. Book.
- Gupta, M. K., Dwivedi, R. K., Sharma, A., Farooq, M., and S, B. R. (2023). Performance evaluation of blockchain platforms. In *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, pages 1–6. DOI: 10.1109/ICICAT57735.2023.10263700.
- Haber, S. and Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology*, 3. DOI: 10.1007/BF00196791.
- Hauser, C. B. and Wesner, S. (2018). Reviewing cloud monitoring: Towards cloud resource profiling. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 678–685. DOI: 10.1109/CLOUD.2018.00093.
- Helebrandt, P., Bellus, M., Ries, M., Kotuliak, I., and Khilenko, V. (2018). Blockchain adoption for monitoring and management of enterprise networks. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1221–1225. IEEE. DOI: 10.1109/IEMCON.2018.8614960.
- Hyperledger (2022a). Hyperledger caliper project. Available at: <https://www.hyperledger.org/projects/caliper>.
- Hyperledger (2022b). Hyperledger whitepaper metrics. Available at: https://www.hyperledger.org/wp-content/uploads/2018/10/HL_Whitepaper_Metrics_PDF_V1.01.
- Hyperledger (2023). Hyperledger fabric docs. Available at: <https://hyperledger-fabric.readthedocs.io/pt/latest/index.html> Accessed: 2023-11-01.
- Ismail, L., Hameed, H., AlShamsi, M., AlHammadi, M., and AlDhanhani, N. (2019). Towards a blockchain deployment at uae university: Performance evaluation and blockchain taxonomy. In *Proceedings of the 2019 International Conference on Blockchain Technology, ICBCCT 2019*, page 30–38, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3320154.3320156.
- Košt’ál, K., Helebrandt, P., Belluš, M., Ries, M., and Kotuliak, I. (2019). Management and monitoring of iot devices using blockchain. *Sensors*, 19(4):856. DOI: 10.3390/s19040856.
- Liang, X., Zhao, Y., Zhang, D., Wu, J., and Zhao, Y. (2021). Sbhps: A high performance consensus algorithm for blockchain. In *2021 International Conference on High Performance Big Data and Intelligent Systems (HPBDIS)*, pages 6–11. DOI: 10.1109/HPBDIS53214.2021.9658348.
- Matteussi, K. J., dos Anjos, J. C. S., Leithardt, V. R. Q., and Geyer, C. F. R. (2022). Performance evaluation analysis of spark streaming backpressure for data-intensive pipelines. *Sensors*, 22(13). DOI: 10.3390/s22134756.
- Melnik, E. and Safronenkova, I. (2023). Ontological approach to the organization of computing in distributed

- monitoring systems with mobile components based on a distributed ledger. In *Interactive Collaborative Robotics: 8th International Conference, ICR 2023, Baku, Azerbaijan, October 25–29, 2023, Proceedings*, page 300–310, Berlin, Heidelberg. Springer-Verlag. DOI: 10.1007/978-3-031-43111-1_27.
- Moschou, K., Theodouli, A., Terzi, S., Votis, K., Tzovaras, D., Karamitros, D., and Diamantopoulos, S. (2020). Performance evaluation of different hyperledger sawtooth transaction processors for blockchain log storage with varying workloads. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 476–481. DOI: 10.1109/Blockchain50366.2020.00069.
- Salama, R., Al-Turjman, F., Altrjman, C., Kumar, S., and Chaudhary, P. (2023). A comprehensive survey of blockchain-powered cybersecurity- a survey. In *2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)*, pages 774–777. DOI: 10.1109/CICTN57981.2023.10141282.
- Samaniego, M. and Deters, R. (2017). Internet of smart things - iost: Using blockchain and clips to make things autonomous. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 9–16. DOI: 10.1109/IEEE.ICCC.2017.9.
- Sharma, P. K., Chen, M., and Park, J. H. (2018). A software defined fog node based distributed blockchain cloud architecture for iot. *IEEE Access*, 6:115–124. DOI: 10.1109/ACCESS.2017.2757955.
- Ward, J. and Barker, A. (2014). Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing*, 3. DOI: 10.1186/s13677-014-0024-2.
- Xu, C., Wang, K., and Guo, M. (2017). Intelligent resource management in blockchain-based cloud data-centers. *IEEE Cloud Computing*, 4(6):50–59. DOI: 10.1109/MCC.2018.1081060.
- Yahaya, S. W., Lotfi, A., Mahmud, M., and Adama, D. A. (2021). A centralised cloud-based monitoring system for older adults in a community. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2439–2443. DOI: 10.1109/SMC52423.2021.9659087.
- Yang, Y., Liu, M., Zhou, Q., Zhou, H., and Wang, R. (2019). A blockchain based data monitoring and sharing approach for smart grids. *IEEE Access*, pages 1–1. DOI: 10.1109/ACCESS.2019.2952687.
- Zhang, L., Li, Y., Qiu, B., Zhang, J., and Liang, W. (2021). Design of communication power centralized remote monitoring system based on big data technology. pages 46–49. DOI: 10.1109/ECIE52353.2021.00017.
- Zou, Y., Peng, T., Wang, G., Luo, E., and Xiong, J. (2023). Blockchain-assisted multi-keyword fuzzy search encryption for secure data sharing. *Journal of Systems Architecture*, 144:102984. DOI: 10.1016/j.sysarc.2023.102984.