IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 4, 2021, accepted May 16, 2021, date of publication May 20, 2021, date of current version May 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3082197

HIDRA: A Distributed Blockchain-Based Architecture for Fog/Edge Computing Environments

CARLOS NÚÑEZ-GÓMEZ[1], BLANCA CAMINERO[2], (Member, IEEE), AND CARMEN CARRIÓN[2], (Member, IEEE)

[1]High-Performance Networks and Architectures Group (RAAP), Albacete Research Institute of Informatics (I3A), University of Castilla-La Mancha, 02071 Albacete, Spain
[2]Department of Computing Systems, University of Castilla-La Mancha, 02071 Albacete, Spain

Corresponding author: Carlos Núñez-Gómez (carlos.nunez@uclm.es)

This work was supported in part by the Spanish Ministry of Science and Innovation under Grant RTI2018-098156-B-C52, and in part by the Spanish State Research Agency under Grant PEJ2018-003001-A.

ABSTRACT Nowadays, the fog computing paradigm is being consolidated as a solution for processing the explosion of data generated by lots of common IoT devices connected to the Internet. In contrast to cloud computing, fog computing achieves efficient data processing without incurring large latencies or data transfers to/from the cloud. The distributed nature of fog computing makes the resource orchestration a challenge. Commonly used centralized solutions are of limited utility in this context. Moreover, fog nodes are usually resource constrained, so the implementation of the management modules should be carefully designed in order not to cause excessive overheads. On the other hand, blockchain has proven its utility beyond cryptocurrencies, to support distributed and reliable information storage. When combined with smart contracts it can provide a distributed computer where all the nodes independently and equally contribute to a common global system state, which must be agreed by consensus. This also provides inherent desirable features, such as immutability and transparency. In this work, a novel architecture called HIDRA is presented, aimed at resource orchestration in fog computing environments based on a Ethereum blockchain implementation. A prototype implementation on a testbed composed of single-board computers has been carried out. Results show the low overhead introduced into the system and its ability to perform a coordinated action among fog nodes without the intervention of any central authority.

INDEX TERMS Blockchain, distributed systems, fog computing, Internet of Things, resource orchestration, smart contracts.

I. INTRODUCTION

Internet of Things (IoT) [1] has recently arisen as a computing paradigm that combines different technologies, such as ubiquitous computing, specific Internet protocols, wireless sensor networks, communication technologies and embedded systems, just to name a few. IoT is causing a great impact in society due to its uncountable applications in industry, health, agriculture, and so on. IoT applications have usually harnessed cloud computing resources in order to extract value from the huge amount of data collected by the ''things''. However, this approach is not feasible for some applications due to the latencies from/to the cloud or the bandwidth wasted in transferring all data to the cloud for their processing. Thus, the concept of *Fog Computing* arises as a means of making storage and computing resources closer to the place where data are generated.

The traditional cloud computing model cannot meet not only the heterogeneous, low delayed, and dense access networks but also the use of various applications for intelligent terminal users. In order to solve this problem, Bonomi *et al.* proposed the fog computing paradigm in 2011 [2]. This model deals with the addition of a fog layer. The main aim is to relocate some cloud core functions to a location close

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

to the network edge. This allows to overcome the drawbacks of cloud computing regarding location awareness, mobility support, and real-time interaction.

Fog computing is the context of this work. In a fog architecture, there exists limited storage and computing resources closer to the end devices. There are different strategies to support fog computing, from integrating computing capabilities within network equipment [3] to establish micro-datacenters close to the things [4] or to harness low cost devices such as Single-Board Computers (SBCs) which offer a great cost/performance ratio (e.g. Raspberry Pi [5]). In fact, a combination of different architectures, operating systems and resource specifications are very usual in current fog and IoT environments [6], [7].

Another issue to consider is related to the fact that the fog infrastructure should be able to support different IoT applications simultaneously. For example, in a smart factory an application for environmental control could coexist with another application for safety and presence control in sensitive places, and with another for workers' health monitoring. Each of these applications should be isolated from the others for security reasons but also in order to provide them with the required Quality of Service (QoS). These are the reasons for the use of light virtualization approaches, namely containers [8], to support and isolate different applications with low overhead. Hence, orchestration tools, such as Kubernetes [9], are usually needed to manage and track the lifecycle of containers, among other tasks. Moreover, some flavours of Kubernetes specifically targeted to Edge/Fog environments have recently appeared [10], [11].

In any case, fog computing is an emerging computing model with new problems and challenges because of its innate features, such as wide distributed, heterogeneous network and fog computing nodes with limited resources. So, the proper management of these features has become one of the main aims in the fog computing research field [12], [13].

On the other hand, *Blockchain* technology [14] has become a trending topic in industry over the last few years. It has overcome its initial application in cryptocurrencies to reach over other application domains, which harness its immutable, transparent and available information storage capability. Moreover, *Smart Contracts* over blockchain [15], [16] can be seen as a distributed computer which is programmed to achieve different goals depending on the application. This type of digital contracts cannot be modified because their behavior itself (i.e., a set of promises) is stored as an irreversible transaction in the blockchain. Smart contracts are executed independently by each node participating in the blockchain. This fact can be harnessed to implement a truly distributed orchestrator for fog nodes, which also provides resiliency in case of node failure or disconnection.

Thus, the main goal of this paper is to describe HIDRA, a *Heterogeneous, Interoperable and DistRibuted Architecture* for the orchestration of applications running in a fog computing environment based on blockchain technology and smart contracts. The HIDRA's main features can be summarized as follows:

- The considered fog environment is local and private, i.e., it is spread over a building or similar area under a unique administrative entity.
- The architecture autonomously manages the resources of the fog nodes (CPU, memory, disk and I/O) and orchestrates containerized applications.
- All fog nodes perform the same management role, as specified by a set of smart contracts, and share a system common state stored in the blockchain. This means that there is no centralized controller or master node having special privileges.
- The system monitors outliers in resource utilization and updates resource allocation according to pre-configured rules.
- Resource management includes tuning reputation mechanisms in order to identify misbehaving nodes.

The structure of the rest of the paper is as follows. First, Section II presents the background and the building blocks of the proposal. Section III provides related works concerning blockchain and resource management in fog computing environments. Section IV introduces the overview of the proposed architecture, and its key components and functional modules. Then, Section V specifies the orchestration techniques of the proposal, and details the functional modules and their associated smart contracts. In Section VI, a proposal validation on a SBC-based prototype is carried out. Section VII discusses the proposal in general terms, evaluating the functionality and possible risks of the developed system. Finally, Section VIII concludes the paper.

II. BACKGROUND AND BUILDING BLOCKS

Mist Computing, *Edge Computing*, *Fog Computing* all support the execution of applications near the data sources. Mist computing enables the IoT devices to process data within themselves whereas Edge computing performs the processing operations at the gateways of IoT devices. Recall that IoT gateways are the interface between IoT specific networks, such as Zigbee or LoraWAN, and the general TCP/IP based Internet. They can have some varying amount of processing and storage power, which can be harnessed for data processing at the edge of the network. Instead, fog computing is a more general concept, and makes use of IoT gateways but also other computing devices within the edge network, such as smart routers, Raspberry Pis, personal computers and even micro-datacenters to process IoT data. A related term which also introduces storage and processing power at or near the edge of the network is *Multiaccess Edge Computing* (MEC). MEC servers are owned by mobile operators and locate them near the base stations so that they can be accessed over the Radio Access Network (RAN). They are mainly targeted to the support of mobile services. Mist, edge, fog and MEC are relatively new computing paradigms, and their evolution processes are still ongoing. So, many researchers and industries

adopt different approaches. A more detailed explanation, which is out of the scope of this paper, is provided in [17].

On the other hand, blockchain technology allows the development of distributed environments that do not need centralized control entities or intermediaries to work properly. This technology is supported by different cryptographic techniques that ensure the immutability and non-repudiability of the data sent by the blockchain participants. Nowadays, blockchain technology is applied in a multitude of use cases, such as cryptocurrency and payment environments (for example, Bitcoin [18]). Other prominent use cases can be notarial services, supply chains or integration with the IoT environment.

Fog computing and blockchain are regarded as being highly feasible because of its broad support for IoT applications [19], [20]. Some more insight into both technologies will be given next.

A. FOG COMPUTING

The fog computing paradigm is a key technology for supporting next-generation Internet uses, such as IoT applications (i.e., smart healthcare, city, agriculture and industry) or the Internet of Vehicles (IoV). The main idea of fog computing is to support computation in proximity to the source of the data. Fog servers in the near-end open the way to efficiently support the traffic flow in the network reducing the network congestion. Additionally, the goal of this architecture is to provide processing capacity and data storage in the edge as an extension of cloud computing. The distributed structure of fog computing supports heterogeneity, provides low latency and improves the Quality of Service (QoS) of real-time and latency-sensitive applications. Nevertheless, the need for efficient resource management, offloading techniques and security issues still remain as open challenges.

Fog resource management performs administrative operations such as deployment, virtualization and monitoring of fog nodes. In addition, fog resource management includes load balancing, auto-scaling and dynamic resource provisioning to assure service reliability and availability [21]. The offloading technique transfers resource-intensive code, or at least a portion of it, to other nearby servers with available resources [22]. This technique allows for example to cope with the computational limitations of the devices, as well as, to improve the energy efficiency of the system.

Based on how the resources are controlled, architectures for resource management in fog systems can be centralized or distributed. Centralized control refers to the use of a single controller that makes decisions about the use of the edge resources. Up to day, most of the research has focused on centralized architectures such as container orchestrators (e.g. Kubernetes, Docker Swarm or Mesos). On the contrary, in a distributed architecture the decision-making process is distributed across the fog nodes. Blockchain-based architectures can be used as a novel support technique to implement distributed control in fog computing systems [21].

B. BLOCKCHAIN

Blockchain technology is capable of storing data in a distributed manner ensuring the data integrity and availability. Underlying peer-to-peer (P2P) networks are used to connect participating nodes together without the need for third parties. A blockchain consists of a data structure that encapsulates data inside transactions. Then, these transactions are sent throughout the P2P network in a broadcast process and grouped into blocks. These blocks are chained together by means of a cryptographic hash function and ordered in time, forming a large database or distributed ledger. A blockchain can be formally defined as a global state machine where transactions are the transitions between states.

There are different types of blockchains such as permissionless and permissioned blockchains [23]. Permissionless blockchains allow any entity around the world to participate, which decentralizes the control and maintenance of the blockchain. In addition, the stored information is public and can be consulted by any entity, which increases transparency. Permissioned blockchains, on the other hand, are focused on local and private use cases where higher performance, control and privacy are required. Only authorized entities can participate in a permissioned blockchain. This means that the control of the blockchain is less distributed, but improves the scalability and network speed [24].

As blockchains broadcast the transactions among all the participants, a consensus algorithm is required to decentralize the blockchain control, ensuring the enforcement of certain consensus rules to the participating nodes. A consensus algorithm allows to obtain a unique blockchain state shared among all participants by establishing which new transactions and blocks will be added to the blockchain. Examples of consensus algorithms are Proof-of-Work (PoW) and Proof-of-Authority (PoA), used in permissionless and permissioned blockchains respectively. In PoW, blockchain nodes compete to generate new blocks and add them to the chain. The competition is based on the computation of expensive cryptographic puzzle operations (generation and search of hashes). This process is known as mining and the entities that execute it are called miners. On the other hand, in PoA, a set of trusted nodes act in turns in a mining rotation schema as block and transaction validators, distributing the responsibility of block creation among all trusted nodes. More information about these and other popular consensus algorithms can be found in [23] and [25].

Ethereum [26] is one of the most popular blockchain platforms. Its main goal is to provide a worldwide computing infrastructure where its cryptocurrency acts as an exchange value for the use of the platform. Ethereum implements its own underlying P2P network which connects all participating nodes on the blockchain. The blockchain transactions (or state transitions) are processed through the Ethereum Virtual Machine (EVM), which is able to execute custom scripts called smart contracts. The smart contracts are developed with high-level Turing-complete languages such as Solidity.

These smart contracts are compiled in bytecode format and stored within transactions. Hence, it is impossible to modify a smart contract once it is deployed. Formally, a smart contract can be defined as a deterministic computer program that is automatically executed based on the inputs received.

One of the most important implementations of the Ethereum platform is Geth [27]. This client is written in the Go programming language and is available for the vast majority of architectures and operating systems (including Android and iOS). Geth enables participating in permissionless blockchains like the Ethereum main network, but also allows to create permissioned blockchains. Nowadays, Geth implements two selectable consensus algorithms for permissioned blockchains: a PoW algorithm called Ethash (used by the Ethereum main network), and a PoA algorithm called Clique.

III. RELATED WORK

The resource management in fog computing, which includes aspects like task offloading, resource allocation or resource scheduling, is a very sensitive topic that has recently attracted many research and solutions [21], [28]. Authors in [29] present a study of the core issues, challenges and future research directions in fog-enabled orchestration for IoT services, which is a key concept within distributed systems. Fog orchestrator challenges are mainly related to optimizing the process of determining and selecting the best IoT appliances and fog components whilst meeting non-functional requirements such as security, network latency, QoS, fault diagnosis and tolerance, etc.

The need for automated procedures to deal with the resource management challenge has been extensively explored using orchestration and hierarchical orchestration architectures [29]–[31]. Some efforts have also considered a choreography approach in which applications and services have direct communication among each other without any central entity coordinating [32].

In [33] a container-based fog computing orchestration architecture is implemented using Kubernetes and extending its functionality with a novel network-aware scheduling. The smart city scenario-based experiments show a reduction in latency compared to the default Kubernetes scheduling. The limitation of this work is that security and availability may be compromised because orchestration is based on a centralised approach.

The inherent distributed structure that lies within both fog and blockchain technologies has motivated the proposal of frameworks that integrate blockchain into fog architectures to deliver enhanced properties to IoT [34], [35]. Most of these solutions deploy blockchain for data management and authentication purposes to solve security and privacy problems and to provide high reliability, as in [36]. Smart contracts can also be deployed between clients and service providers to validate SLA compliance for on-demand resource usage, as in [37].

For example, the FogBus [38] framework is an end-to-end IoT-Fog-Cloud integrated architecture that applies blockchain as a distributed database. FogBus uses authentication and encryption techniques to secure operations on sensitive data. Author's evaluations reflect that adding security features to the proposed architecture increases the computational cost of the system affecting the service delivery latency, energy and network usage.

An auction-based market model for efficient computing resource allocation in a cloud or fog computing environment is presented in [39]. The auction mechanism is designed considering the blockchain allocative externalities and a limited amount of local computing resources. Two bidding options were developed, the constant-demand scheme and the multi-demand scheme. For the multi-demand scheme, the authors design an approximate algorithm that solves the NP problem of maximizing the return of all participants.

In [40], a blockchain-based fog computing resource contribution model is proposed. The model applies a reward and punishment incentive mechanism to encourage the fog nodes to actively contribute their resources. A differential game method was employed to simulate the optimal resource contribution strategy of fog nodes. The proposed model considers a satisfaction degree (task completion degree) as an evaluation index for service provided by fog computing providers.

Another blockchain-based framework for IoT, called EdgeChain, is described in [41]. The framework integrates a permissioned blockchain and a currency system to design a credit-based resource management system. Smart contracts are used to regulate the IoT devices' behavior and enforce the predefined management rules and policies. Moreover, all the IoT activities are recorded into blockchain for secure data logging and auditing.

DualFog-IoT [42] is a proposal to integrate support for IoT applications that use blockchain. The authors identify three types of applications, namely, real-time (RT), non-real-time (NRT) and delay tolerant blockchain (DTB) applications. Their proposal includes a separate fog infrastructure to process DTB requests, so that the mining processes do not interfere with the performance of RT and NRT requests, which are processed in a conventional fog infrastructure.

In BlockEdge [43], the authors propose to use blockchain as a means for auditing the different processes involved in an industrial IoT application. They evaluate their proposal on the use case of the process of building a smart house, where different agents are involved (contractors, manufacturers…) and over different industrial processes (wood harvesting, logs and brick manufacturing, transportation…). Thus, their proposal does not deal with the use of blockchain as a founding layer for the fog infrastructure.

Finally, in [44], authors present the integration of the MultiChain blockchain framework into a low-cost fog computing environment based on Docker and Docker Swarm. The fog layer is composed of controller nodes and gateways enhanced by SDN functionalities that are integrated into a hierarchically structured architecture.

HIDRA differs from the related work mentioned above in the consideration of blockchain as the first-class technology to cope up with a distributed, fault-tolerant, secure and auditable fog architecture applying orchestration techniques. Thus, blockchain and smart contracts are the key technologies to manage fog nodes fostering the integration of containerized applications through generic APIs. Moreover, in most of the proposals, fog/edge nodes are assumed to have significantly more computing and storage resources than end devices. In contrast, HIDRA's implementation on constrained devices has demonstrated to be feasible as it will detailed later. To sum up, Table 1 summarizes the main features of HIDRA, compared with the most similar approaches found in the literature.

IV. PROPOSED ARCHITECTURE

The main goal of the HIDRA architecture is the smart, secure and distributed resource management in fog computing environments. The architecture has been designed in a generic way and is also application agnostic, which allows for an easy adaptation to most fog environments. The architecture forms a cluster of heterogeneous devices with a localized geographical location, covering an action area within a building, a factory or an university campus. The cluster is considered as private outside the action area and public inside it. For example, only the residents in a smart building can manage the cluster resources and access the services it provides. Besides, this allows for a physical control of the system.

The proposal uses blockchain and light OS virtualization technologies to orchestrate the containerized applications of the cluster following a distributed approach. Each cluster node runs a blockchain service to register and share its state, allowing all the nodes to be aware of the actual state of the cluster. This information is key to achieve an intelligent and distributed resource allocation in the system.

To that end, every node continually monitors its own resources and establishes a set of rules and policies depending on the limitations of the node and the network. For example, a rule can set an upper limit on the percentage of CPU usage. When a node breaches the rule, it shares its state with every other, by sending it through blockchain transactions. Then, the other nodes can reply to this event by sending their current states, which allows to compare the different node states and conclude which node has more available resources or which one is more saturated. All of this is carried out in a completely distributed way, thanks to the deployed smart contracts which will be detailed later.

A. SYSTEM OVERVIEW

Fig. 1 depicts a generic IoT architecture composed by three layers: the *Devices Layer* which hosts the IoT devices, the *Distributed Fog Layer* where HIDRA is focused, and finally, the *Cloud Layer* where mini-datacenters and cloud providers are located.

The Distributed Fog Layer is an extension of the Cloud Layer closer to the final IoT devices, which belong to the

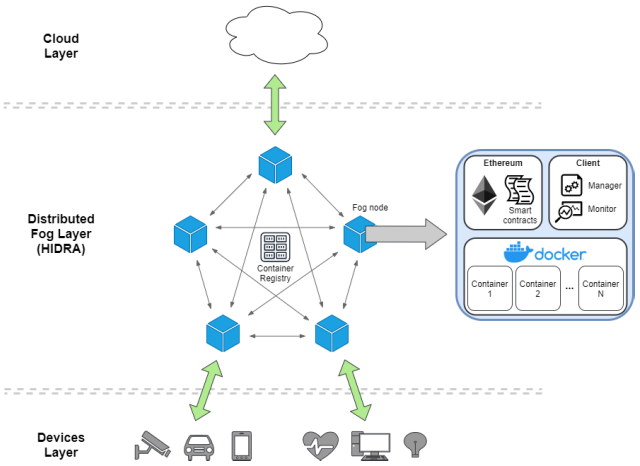


FIGURE 1. System overview.

Devices Layer. The Distributed Fog Layer is placed at the edge of the network and supports the end devices, by offering them services in a more reliable, scalable and fast way. The architecture is well-suited for a huge variety of use cases [34]. The types of end devices in the Devices Layer are widely heterogeneous, ranging from laptops or smartphones connected to a campus network to IoT devices with scarce resources such as temperature sensors or gyroscopes in smart cities.

The nodes belonging to the Distributed Fog Layer are interconnected by means of the *blockchain technology* and an underlying P2P network. These nodes constantly share and request information on the state of the cluster, i.e., the features of every node, the actual resource usage and the applications/containers being run. HIDRA harnesses the Ethereum blockchain [26] to distribute the cluster's data and control. Moreover, the use of *smart contracts* (see Section IV-B2) enables the establishment of a series of terms that every node must equally comply with. On the other hand, every fog node uses a light OS virtualization software to automate the deployment of containerized applications. This fact reduces operational costs and enables a multi-tenancy system. Docker has been chosen for this purpose as it is by far the most mature and adopted OS-level virtualization platform [45].

By leveraging blockchain and smart contracts, the network keeps a distributed container registry that stores the historic of deployed applications/containers and the applications/containers that are actually running. For every container a set of metadata is registered, such as its type, IP address and port, and the resource specifications needed to run it. Note that containers will also have associated the identifier of the application to which it belongs. The blockchain and the light virtualization service are interconnected through a software client that also acts as a daemon to monitor the node state. This refers to the Manager and Monitor modules shown in Fig. 1. The first one deals with the blockchain interaction, the control of the light virtualization service executed on the fog node and the management of events concerning the state of the cluster nodes. The second one monitors the state of the node and sends alerts to the Manager in case certain rules or

TABLE 1. Main features of approaches similar to HIDRA.

Approach	Fog/Edge Infrastructure	Fog/Edge Management	Blockchain Purpose	Blockchain Type	Blockchain Placement	Consensus Algorithm	Smart Contract-Based Management	Fog/Edge Resource Isolation	Incentive Mechanism	Evaluation Platform	Evaluation Metrics
Tuli et al. [38]	Multiple roles (broker, computing and repository nodes)	Master and worker nodes	Ensure data integrity in Edge-Fog-Cloud environments	Private custom blockchain	Fog layer	PoW	No	Virtual machines and containers	-	Testbed (sleep apnea analysis real-world application)	CPU and memory usages, number of requests, latency, network usage, power consumption
Wang et al. [40]	Fog nodes/servers	Cloud layer-assisted	Provide blockchain incentive model characteristics	Coalition chain	Cloud layer (full nodes) and fog layer (light nodes)	DPoS	No	-	Rewards and penalties	Simulation	Optimal resource contribution strategy, satisfaction level and incomes of fog nodes
Pan et al. [41]	Edge servers	EdgeChain framework	Manage resources using smart contracts, secure data logging and auditing in Edge-IoT environments	Permissioned	Edge servers	PoW	Yes	Virtual machines	Rewards and penalties	Testbed (EdgeChain prototype)	CPU, memory and disk usages, block and transaction delays, block sizes, acceptance rates
Memon et al. [42]	Fog Cluster (FCC) and Fog Mining Cluster (FMC)	Centralized (access point-based)	Distributed ledger for delay-tolerant blockchain applications	Permissionless	Fog Mining Cluster (FMC)	PoW	No	-	-	Simulation	Drop rates, resource utilization, response times, number of requests, throughput
Kumar et al. [43]	Edge nodes near IoT clusters (local networks) and fog nodes (global network)	Edge and fog nodes	Trusted data sharing, authentication and access control in IIoT environments	Permissioned (edge nodes) and permissionless (fog nodes)	Edge and fog nodes	-	Yes	Virtual machines and containers	-	Simulation	Latency, power consumption, network usage
Cech et al. [44]	Controller nodes and fog gateways	Centralized	Enhancement of HCL-BaFog fog architecture (secure sensor data storing and sharing)	Permissioned	Fog nodes	Round-robin scheme	No	Light virtualization (Docker containers)	-	Testbed (SBC-based prototype)	Transactions per second, latency
HIDRA	Interconnected fog nodes	Fog nodes (distributed and consensual)	Distributed resource management, device authentication and trusted data sharing in fog environments	Permissioned	Fog nodes	PoA	Yes	Light virtualization (Docker containers)	Distributed reputation system based on rewards and penalties	Testbed (HIDRA prototype)	CPU, memory and disk usages, cluster events times, block sizes, power consumption

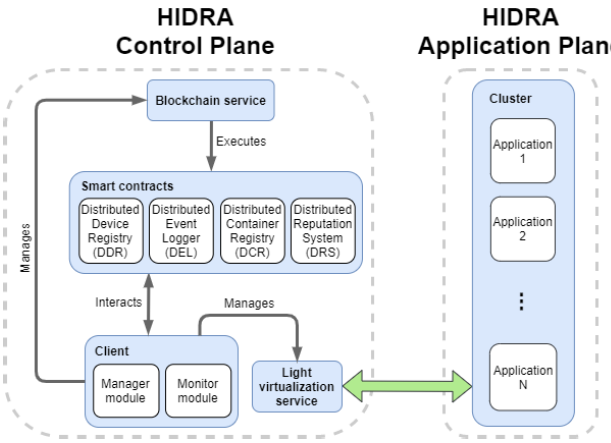


FIGURE 2. HIDRA control and application planes.

policies are breached. As an example, the Monitor can get the list of running processes or the utilization percentage of a node's resources or a specific container.

Up to now, the HIDRA system has been presented from an architectural and structural viewpoint, i.e., its different layers and the structure of the fog nodes are shown. Moreover, the Distributed Fog Layer is logically divided in a Control Plane and an Application Plane. Fig. 2 depicts the modules that compose each plane and their interactions. The Control Plane view encompasses the components and services that manage the cluster resources and orchestrate its containers, being responsible for keeping an efficient well-balanced use of physical resources. In general, smart contracts manage the event flow of the cluster and keep an up-to-date registry of fog nodes and IoT devices as well as cluster containers. The Control Plane client takes care of Manager and Monitor tasks mentioned above. Besides, the client acts as a bridge interface between the light virtualization service and the smart contracts executed by the blockchain service. On the other hand, the Application Plane makes use of containerized applications to better isolate the tenants resources and simplify the physical resources orchestration. Containers provide isolation to the IoT applications from software deployment and physical hardware which ensures portability. A container runtime that supports a standard interface is chosen. All the containers are identified for authenticating and controlling the access to others resources. This plane is composed mainly of two kinds of applications: ones that are focused on getting data from different sensor sources that can be either static (for example, air pollution of a zone) or dynamic (velocity of a vehicle) and others that provide services to the users (for example, offloading computational processing to the cloud). It must be highlighted that the high amount of data produced by, for example, IoT streaming applications will be stored in off-chain storage and only a proof of the data (i.e., a hash value) will be stored in the blockchain.

B. KEY COMPONENTS AND FUNCTIONAL MODULES

In this section, the overall key components and functional modules of the proposed resource management and orchestration system are introduced. More precisely, this section provides some insight into the blockchain service, the proposed functional modules, the fog node client and the light virtualization service included in the Control Plane as depicted in Fig. 2.

1) BLOCKCHAIN SERVICE

HIDRA uses the Ethereum platform to distribute the control and management of the resources and services within the cluster, by harnessing its blockchain technology and its underlying P2P layer. Every fog node runs an Ethereum client that is in charge of sending, receiving and storing data and control events in the cluster (i.e., the generated transactions). The Ethereum client is also in charge of running the smart contracts by using the EVM. Thanks to the underlying P2P layer in Ethereum, a fog node can share its state, synchronize the global state of the cluster, reach an agreement among the nodes and take common decisions on resource and application management from the Application Plane.

In order to manage the Fog-IoT computing environments, the use of permissioned blockchains is proposed. In addition, the Proof-of-Authority algorithm is used to achieve consensus among the nodes. Due to the device heterogeneity of the proposed Fog-IoT architecture, it is possible that some devices do not have enough capacity to run all the tasks associated with the blockchain client. Thus, our system allows nodes to execute full and light blockchain services, mimicking Ethereum's full and light nodes, depending on device constraints. More precisely, a *full blockchain service* stores all the data of the blockchain and participates in consensus events, while a *light blockchain service* can only partially store the blockchain. Because of this, constrained devices need to interact with a full blockchain service to send transactions or retrieve information from the blockchain.

2) FUNCTIONAL MODULES

The smart contracts located in the Control Plane (see Fig. 2) implement the logics of the four functional modules that control resource management and application orchestration in the cluster. Note that the smart contracts deployed in the blockchain enable the distribution of these functionalities among all the participant nodes. These smart contracts are stored in the blockchain, and because of that, they can be independently accessed and run by every fog node (directly or indirectly through a full blockchain service). Moreover, a smart contract stores its state permanently and isolated from other smart contracts using the same blockchain. This allows all the nodes to share a global state.

When a smart contract is deployed, an Ethereum address in the blockchain is assigned to it. This address is used as a smart contract identificator so that the fog nodes can get it back and run it. Smart contracts cannot be edited nor their storage migrated once deployed, because this would imply the deployment of a new smart contract in another different Ethereum address. For this reason, the smart contracts from the functional modules have been designed and implemented

in a universal way, so that no modifications are needed in case the cluster requirements change in the future. This also enables the proposed architecture to adapt to a wider variety of use cases in fog computing environments. The different functional modules and their associated smart contracts are introduced next. They are more thoroughly described in Section V-B.

- **Distributed Device Registry (DDR)**. A device can take part in the cluster only if it has been previously registered. The DDR smart contract stores a list of registered devices, which can be fog nodes or IoT devices linked to those fog nodes. For each registered device, an Ethereum address previously generated is stored. Some metadata for auditing, monitoring and orchestrating purposes are also stored (for example, device specifications and its configuration within the cluster). According to the device type, the process to store these metadata changes. On the one hand, when a fog node is registered in the cluster, a new isolated smart contract is deployed. This allows fog nodes to create an identity card to control and isolate their own metadata, which increases the security and transparency of the cluster. Moreover, isolated smart contracts make it easy to update other Control Plane smart contracts because device metadata are distributed, i.e., they are not stored in an unique controller smart contract. Note that a fog node smart contract implements the necessary procedures to manage its metadata (create, update and delete). On the other hand, there are IoT devices with limited resources unable to even run an Ethereum light node, which prevents them from interacting directly with the blockchain. In such cases, the registration process is carried out by a fog node on behalf of the constrained device. The fog node generates an Ethereum address for the IoT device and registers the metadata of the new device in its own smart contract.
- **Distributed Event Logger (DEL)**. Every fog node in the cluster monitors its own state and applies specific custom rules regarding resource utilization. A fog node that detects the breaching of a rule can issue an alert about it as an event entity to every other node in the cluster. From that moment, that node turns into an applicant fog node. An event can be used to request, reallocate or release resources as containers, to send information about a fog node (specifications or current state) or to notify changes in cluster configuration. The DEL module smart contract manages the storage and exchange of such events. When a new event is registered, each cluster fog node receives it and sends a transaction reply to this event containing its own metadata. For example, an overloaded node can send a container migration event and the other nodes can reply with their current resource usage. This enables every node to evaluate the state of the other nodes and select the most adequate node to host the container in a consensual way. In order to decentralize the process and avoid malicious elections, the fog nodes register their election in the DEL smart contract (i.e., they vote a node). Then, the most voted node, called solver node, carries out the resource management tasks and changes the state of the cluster accordingly.
- **Distributed Container Registry (DCR)**. The fog nodes that take part in the cluster are able to manage their own resources and to autonomously orchestrate the applications run in the Application Plane (see Fig. 2). A fog node can deploy applications into containers by means of a light virtualization service. The DCR module implements a smart contract that stores the global state of the containers running in the cluster. Every container is modelled as a register that stores some container instance metadata. This enables a backlog of containers, separating old ones from those that are currently running. On the other hand, the DCR module registers the desired cluster state (i.e., desired containers, number of replicas, desired hosts to run these containers and so on), which allows container orchestration. This contributes to the self-healing of the running applications. For example, if a fog node fails or reboots, as soon as it recovers its normal state, it accesses the DCR and retrieves the containers which must be executed.
- **Distributed Reputation System (DRS)**. In order to measure the behavior of the cluster devices and their degree of both trust and participation, the proposed system integrates a reward and penalty mechanism. The system links a dynamic reputation value to registered devices which participate in the cluster management and orchestration tasks. The reputation value of a fog node depends on its operation over time. The proposed mechanism increases or decreases fog node reputation depending on the basic actions they carry out. A fog node will increase its reputation value if it attends its tasks regarding cluster management and orchestration (for example, it runs the container instances assigned to it, takes part in the process of voting for solver nodes, or provides additional resources to the cluster). On the other hand, a fog node will decrease its reputation when under-performing on the Quality of Experience (QoE) expected or when a cost is generated due to using the cluster resources (i.e., when executing actions such as new application deployment or container migration requests). The DRS module smart contract implements the proposed reward and penalty mechanism. This smart contract contains an immutable list of reputable cluster actions and implements the logic required to apply reputation variations to cluster nodes depending on the action they take. The reputation system aims to reflect the past behavior of fog nodes and avoid misbehaviors in the cluster such as cluster event flooding or resource wasting (e.g. container flooding). Furthermore, the reputation mechanism allows quantifying other factors such as availability or computational power of the cluster nodes. Hence, the DRS module executes a trusting algorithm that decreases or stops

malicious behaviors. So, the reputation system is also used to enhance access control security.

3) FOG NODE CLIENT

Besides the blockchain service, every fog node in the cluster runs a software client that enables the intercommunication between all the Control Plane functional modules. This client enables the distribution of cluster control and logic in parts that cooperate towards a common goal. In this way, master management nodes and single point of failures are avoided. To this end, every client implements the Manager and Monitor modules (see Fig. 1). Apart from interconnecting the cluster functional modules, the clients undertake the management, monitoring and rule and policy enforcement of the fog node where they run. The client also acts as a proxy for the constrained IoT devices that cannot run a blockchain service to interact with the system. The fog node clients implement the Manager module, that interacts with the Ethereum blockchain service, with the deployed smart contracts and with the light virtualization service. This translates into different interfaces that manage requests and responses among functional modules.

In the event of adding a new fog node to the cluster, the client generates an Ethereum address that allows it to interact with the blockchain service and also unambiguously identifies it within the cluster. Then, the client registers the fog node in the cluster by sending a transaction to the DDR module smart contract. The process to add an IoT device to the cluster is analogous, except for the fact that there exists a proxy fog node that performs the register process on behalf of the IoT device. Note that every fog node client keeps a listing of Ethereum addresses that includes its own address and address for every associated IoT device. Once a fog node has been registered, it is able to interact with every functional module and management smart contract, in particular: (1) thanks to the DDR module, the client can modify the metadata stored in its isolated smart contract, add new IoT devices and manage the existing ones associated to the fog node; (2) the client can send and receive cluster management events using the DEL module; (3) making use of the DCR module, the client can retrieve the cluster container state and the information on every running container, and register or delete containers; and finally, (4) the client can query its current reputation as well as the reputation of other fog nodes in the DRS, which makes it possible taking orchestration decisions.

On the other hand, the fog node client implements the Monitor module, which is in charge of monitoring resource usage. The client constantly monitors the fog node basic specifications, for example CPU and memory usages. These monitoring data are called *dynamic metadata*. Monitorization is carried out at a global level, obtaining the global usage of cluster nodes. Furthermore, the client would be capable of monitoring other features such as the resource usage of running containers or node processes and network connections, enhancing the cluster control and management. Note that the current work is aimed at detailing the architecture and structure of the orchestration system, so only the basic specifications are included in the proposal.

Besides monitoring dynamic metadata, the client Monitor module gets and manages some *static metadata* necessary for the correct operation of the DDR and DCR smart contracts, i.e., metadata associated to the cluster devices (for example, the maximum amount of resources in a fog node or the geolocalization of a device) and the instance container metadata (the maximum amount of resources allocated to the container, service port, and so on). Static and dynamic metadata are used by the client Manager to enforce management and orchestration decision making. The Manager performs enforcement actions taking into account 1) the usage data obtained in every monitoring interval, 2) the rules defined in the fog node, and 3) the static and dynamic metadata registered by the other cluster nodes. According to the three former data inputs, the Manager is able to detect particular usage values, select the best/worst nodes in terms of current resource usage or reputation, and finally, identify problematic running containers and orchestrate them.

4) LIGHT VIRTUALIZATION SERVICE

Once the fog nodes can interact among them, monitor their own state and take orchestration decisions, it is necessary an application virtualization environment that enables the management of software containers according to the decisions jointly taken by the nodes of the cluster. Light virtualization approaches have been demonstrated to introduce negligible overhead in any environment, and specifically on SBCs [8]. The proposed system harnesses the open source light virtualization Docker tool to aisle the applications running in the fog nodes and ease their management and orchestration. Docker has been chosen as it is by far the most mature and adopted OS-level virtualization platform. Using Docker, the cluster manages the resources used by the applications (containers) and is able to limit or allocate resources according to the specifications and requirements of fog nodes and running applications. Besides, Docker enhances cluster security and modularity and enables a consistent environment for software execution, by reducing compatibility issues among software versions, architectures and operating systems.

Every fog node runs a Docker service and a set of containers that correspond to one or more applications deployed in the cluster. The fog node client controls the Docker service through the Docker Engine HTTP API. In this way, the client can create, edit and delete containers, and access all the information from the Docker service as well as the state of the containers. For example, the client is able to obtain the amount of resources allocated to every container, and can modify it according to the requirements of the running applications. Note that the client also manages the Docker images needed by the application containers. When deploying a new container, the client checks if the Docker image exists in the fog node. The client permits the configuration of an image repository to download missing images. By default, the proposed system uses the Docker Hub repository.

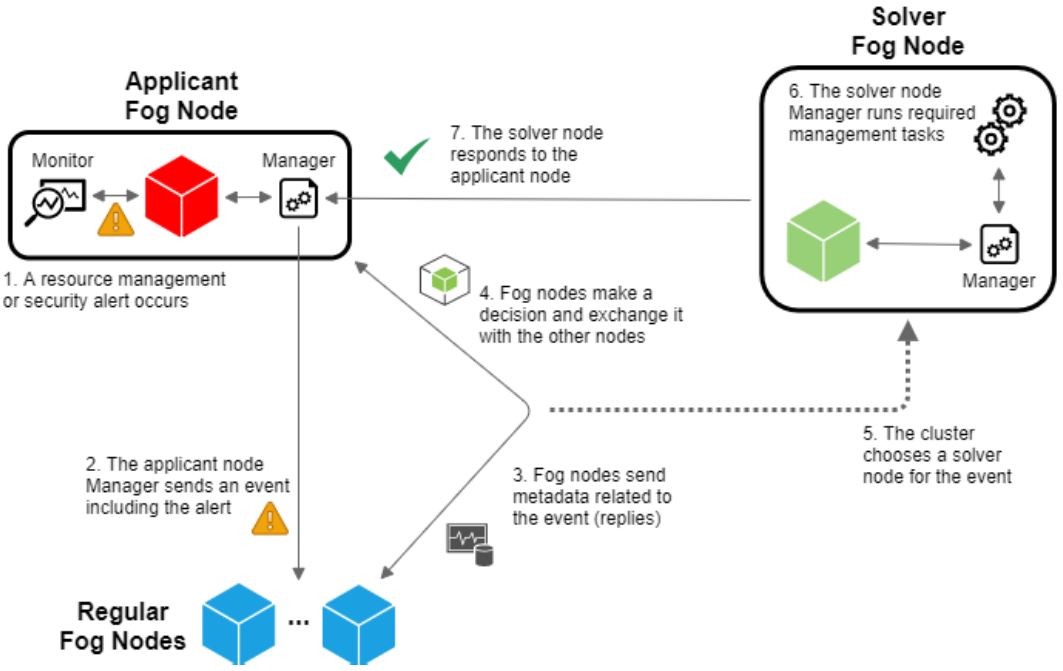


FIGURE 3. Main workflow protocol.

The use of Docker together with the DCR module enables a shared and global container state that any fog node in the cluster can query. The client acts as an intermediate between these functional modules, which allows the modification of the local container state in every fog node depending on the cluster DCR state. On the other hand, the DEL module enables sending events to the other fog nodes, including metadata related to the containers. These metadata specify the action requested by the applicant node and identify the involved containers (e.g., a container migration event that includes the identifier of the container instance in the DCR module, so that the solver node selected by the cluster can replicate the container within its own Docker service).

V. ORCHESTRATION DETAILS

The previous section has presented an overview of the proposed system defining its layered architecture, the structural key components and the functional modules. This section delves into the orchestration details and design requirements of HIDRA, detailing its workflows and the interaction between modules. The section is divided as follows. First, Section V-A defines the main workflow protocol designed to synchronize cluster nodes and make autonomous and consensual management decisions. Section V-B describes each of the aforementioned functional modules detailing workflows, basic actions and metadata. Finally, Section V-C specifies the software client design requirements in the process of monitoring and orchestrating resources.

A. MAIN WORKFLOW PROTOCOL

The interconnection and synchronization of the fog nodes is performed thanks to the proposed main workflow protocol that we will detail next. This protocol is triggered whenever a fog node is no longer in the steady state and discovers an issue. Fig. 3 outlines the main components and steps of the workflow protocol. Hence, the main workflow protocol relates the different states of a fog node, and specifies the actions carried out by the nodes. It highlighted that from the control point of view of our system, the fog nodes can be in different states such as applicant mode, solver mode or regular mode. So, the regular mode matches with the steady state of the fog node. A fog node remains in the applicant mode when the Monitor module finds a problem related to non-compliance with the specific rules of the node. Finally, the solver mode is fixed in a node when it is chosen for the regular nodes as the most suitable node to provide a solution to an applicant node.

As shown in Fig. 3, the cluster state update starts automatically as soon as a fog node switches to applicant mode. From this point on, the applicant node and the regular nodes (the others) work together to solve the issue presented by the applicant node. To reach this goal, the cluster nodes exchange small data structures or objects containing the dynamic metadata required in the process. For more information about the structure of the rules and exchanged objects see Section V-B2. The detailed steps of the protocol are as follows:

1) The Monitor module of a fog node discovers a resource management or security issue checking its pre-configured custom rules. Depending on the pre-configured verbosity of the triggered rule, the Monitor generates and sends an alert to the Manager module.
2) The Manager module encapsulates the alert and the current state of the node's resources in an event object. Then, the event object is sent to the regular fog nodes

through a broadcast process. Henceforth, the fog node becomes an applicant node requesting the cluster collaboration to solve its issue.

3) When a regular fog node receives an event object, it sends a reply object to the cluster that contains some dynamic metadata about its current resource state. All replies are assigned to the applicant node event.
4) Once the state of all the nodes has been provided, each fog node (including the applicant node) takes into account the current states of the cluster nodes and runs a heuristic algorithm to select the most suitable node to become a solver node to assist the applicant node. Then, the cluster nodes exchange the decisions made. This information is sent in the form of votes. Like reply objects, vote objects are also assigned to the applicant node event. Choosing the solver node in a consensual way allows blocking dishonest decisions of nodes that do not follow the cluster rules or try to harm other fog nodes.
5) Once the required votes have been exchanged, the cluster notifies the fog node chosen to solve the applicant node event. Henceforth, this fog node becomes the solver node.
6) The solver node Manager executes locally the management and orchestration tasks required to fix the applicant node issue. Depending on the issue type and the tasks executed, the solver node could update the cluster global state.
7) The solver node notifies the applicant node that the issue has been solved. At this point, the event closes and the applicant node can execute other management ending tasks, if required. Note that only the consensually chosen solver node can close the event and notify the applicant node.

The main workflow protocol details the states and relationships between the cluster fog nodes in a conceptual way without specifying the use of specific technologies. This will allow us to explore additional building blocks in the future beyond those used in the architecture proposed in this paper. As specified in Section IV-B2, in order to carry out resource management and application orchestration, four functional modules based on blockchain technology and smart contracts are defined. The DEL module implements the proposed main workflow protocol. Moreover, in the proposed implementation, the cluster fog nodes are interconnected through Ethereum's underlying P2P network and use its blockchain and the EVM to execute smart contracts and store and exchange information by sending transactions. Event, reply and vote objects, named respectively in steps 2, 4 and 5 of the protocol, are managed using the DEL smart contract. On the one hand, the Manager module of each fog node is responsible for generating and sending transactions to the DEL smart contract containing the required metadata. To this end, the Ethereum addresses of each device are used. On the other hand, the EVM allows the recording of logs related to events that happen in smart contracts. These logs are stored parallel to blocks and transactions as historical data. Moreover, it can be programmed to send them under certain conditions. Ethereum logs are considered as asynchronous triggers and data containers. Sending logs, the DEL smart contract can notify and send metadata back to fog nodes when a cluster event happens. Thus, the DEL smart contract is able to notify the other cluster nodes when an applicant node sends an even object, when the reply or vote objects are collected, or when the cluster global state changes (for example, when a solver node closes an event or a new container instance is registered).

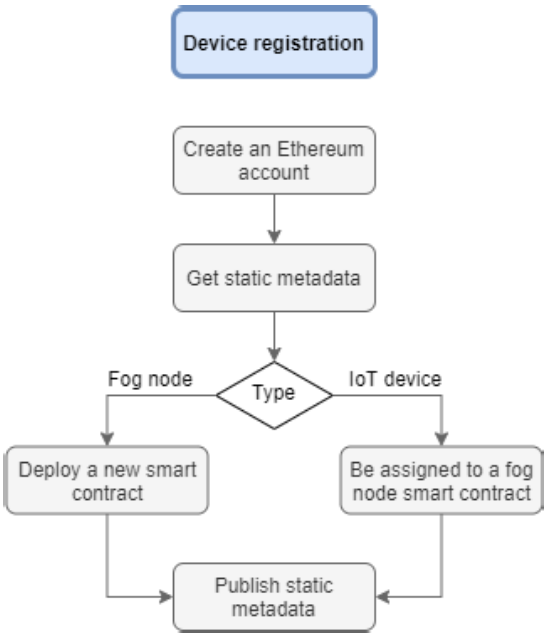


FIGURE 4. DDR workflow.

B. FUNCTIONAL MODULES DETAILS

This section details the workflows provided by the functional modules and the management of events and metadata carried out by the smart contracts within the HIDRA architecture.

1) DISTRIBUTED DEVICE REGISTRY (DDR)

The DDR module is responsible for registering new cluster devices. When registering a device, this module stores the static metadata required to audit and monitor the device and to orchestrate its hosted applications. These metadata contain the device specifications, such as the CPU type (architecture, number of cores and maximum frequency), the total amount of memory and disk storage, and additional metadata such as the running operating system. Furthermore, the static metadata contain the device configuration within the cluster (device IP, device geolocation, and so on).

Fig. 4 shows the registration process of fog nodes and IoT devices. Regardless of the device type, a fog node client must generate an Ethereum address to interact with the blockchain service and must obtain the new device static metadata to subsequently publish them in the cluster. In the event of an IoT device registration, a fog node can do the registration

TABLE 2. DEL objects.

Object	Metadata	Data Type	Definition
Event	*type*	String	Serialized data structure representing the event type. It encapsulates optional metadata such as the problematic resource type, the task to be executed by the solver node or some data about one or more containers. The metadata included depends on the event type.
	sender	Ethereum address	Applicant node Ethereum address.
	solver	Ethereum address	Solver node Ethereum address.
	replies	Array	Set of reply objects sent by fog nodes in response to a cluster event.
Reply	*replier*	Ethereum address	Replier node Ethereum address.
	nodeState	String	Serialized data structure containing the dynamic metadata obtained by the Monitor module of the replier node.
	voters	Array	Set of fog node addresses that have chosen this replier as the event solver.

process on behalf of a constrained IoT device (acting as a proxy node). This device does not have enough resources to run the blockchain service and the developed client. Hence, the constrained devices need to be controlled by a parent fog node. So, a parent fog node is in charge of sending Ethereum transactions on behalf of their IoT device during the entire system lifecycle.

Once the Ethereum address is generated, the fog node obtains the new device static metadata, either through its Monitor module, in case of fog node registrations, or through other ways such as MQTT protocol requests [46], [47] or manual settings in case of IoT device registrations. Note that IoT devices are only registered for identification and geolocation purposes. The next step in the workflow is the device registration using the generated Ethereum address. If the requested action consists of registering a fog node, the fog node client deploys a new isolated smart contract on the blockchain. On the other hand, if an IoT device has to be registered, the fog node assigns the IoT device to its own smart contract. Finally, the obtained metadata are published on the cluster by storing them in the fog node smart contract. In this way, the metadata can later be retrieved by other cluster nodes.

The DDR module shares with the other functional modules an unique management smart contract that stores and controls the cluster global state. This smart contract stores the list of registered fog nodes in the form of Ethereum addresses. Note that both the addresses of the deployed node smart contracts and the identification addresses generated by the fog nodes are stored in the management smart contract. Regarding the DDR module, the management smart contract only keeps the list of registered nodes and implements the logic to perform the cluster registrations. On the other hand, each smart contract deployed by the fog nodes implements the logic to publish and update the fog node static metadata and to register IoT devices.

2) DISTRIBUTED EVENT LOGGER (DEL)

The main workflow protocol described in Section V-A corresponds to the DEL module workflow. The DEL module implements the main workflow protocol using the Ethereum platform along with its smart contracts and log asynchronous triggers. Once a fog node has been registered, it begins to monitor its own state (regular mode) and interact with the other cluster nodes. Monitoring tasks take place in the client Monitor module at constant time intervals.

Dynamic metadata are involved in the rule checking process and the solver selection process. At each monitoring interval, the client Monitor obtains the metadata related to the fog node current state and compares it with the preconfigured rules. These dynamic metadata include the CPU, memory and disk utilizations, and in-out network packet statistics. On the other hand, fog nodes need to know the current state of the other cluster nodes to make their decisions. For this reason, client Managers reply to event objects sending the dynamic metadata obtained by the client Monitors. Note that the solver selection process is the same for all cluster nodes and, through the DEL voting, nodes are able to detect and discard dishonest votes.

When the client Monitor detects a triggered rule, it prepares an alert and sends it to the client Manager module. The Manager is responsible for encapsulating the alert in an event object and sending it to the cluster nodes through a blockchain transaction. Table 2 details the format of both event and reply objects managed by the client Manager. Each event specifies its type dynamically, i.e., the event type is defined based on the metadata included in a serialized data structure within the event object. Depending on the event type, the cluster might not need to send reply objects or choose a solver. Fog nodes send events in a broadcast way through the blockchain network, but it is also possible to send unicast events focusing on a single cluster node. In this way, the DEL workflow is not only limited to solve applicant node requests, but also enables to notify other types of basic actions to be performed by the cluster nodes. These actions can be grouped into the following cluster event types:

- **Container deploy events**. Fog nodes are able to request the deployment of a new container in the cluster. This is a broadcast process, being necessary to exchange both reply and vote objects to select a solver node to host the container. This process allows load balancing across the cluster.
- **Container migration events**. An applicant fog node sends a request for help to the cluster. The node has

TABLE 3. Container instance metadata.

Metadata	Data Type	Definition
appId	Uint	Application identifier to which the container belongs.
holder	Ethereum address	Ethereum address of the applicant node that owns the container.
host	Ethereum address	Ethereum address of the solver node hosting the container.
type	String	Serialized data structure representing the container type. It contains orchestration metadata such as the container impact on the cluster (see Section V-C for more details).
imageTag	String	Docker image name and version tag.
cpuLimit	Float	Maximum number of CPUs to use by the container.
memLimit	Uint	Maximum container memory in bytes.
volumes	Array	Set of Docker volumes to bind to the container.
ports	Array	Set of network ports to bind to the container.

discovered a resource issue and needs to migrate one of its hosted containers to another cluster node. As in the previous event type, fog nodes exchange reply and vote objects and choose a solver node.

- **Container management events**. A fog node requests that another cluster node perform a management task on one of its containers such as reconfigure the container parameters or restart or delete the container. This event type is a unicast process, only the fog node hosting the container is notified (see Section V-B3 for more details). Note that this event type enables the vertical scaling of containers.
- **Rule management events**. Each fog node contributes a specific amount of resources to the cluster according to its own defined rules. Assuming that these resources are not always the same as the physical capacity of the node, a rule management event could be used to perform a rule reconfiguration. Hence, this type of broadcast event leads to an increase or decrease in the amount of resources that a fog solver node contributes to the cluster.

As in the DDR module, the management smart contract implements the required logic to carry out the DEL workflow. This smart contract contains the procedures for sending DEL objects, selecting solver nodes and solving events. Each procedure implements the log asynchronous trigger that notify cluster nodes when a cluster basic action is executed. To perform unicast communications, client Managers implement filters to discard some of the messages notified by the management smart contract. Thus, fog nodes are able to discard the messages that are not intended for them. Note that all Table 2) metadata are stored in the blockchain through the management smart contract, allowing the information to be accessed later, which increases the auditability of the process.

3) DISTRIBUTED CONTAINER REGISTRY (DCR)

The DCR module manages the information of all containers executed in the cluster. Fog nodes access the DCR to query the container instances executed by other fog nodes, and to update the state of their own containers. In order to describe the DCR workflow, a container deploy event is detailed below. Fig. 5 shows the DCR workflow steps to deploy a new container in the cluster, where the dashed lines refer to the nodes that perform every step. In the DCR module context, the applicant fog node that wants to deploy a container is called holder and the solver node that finally runs the container is called host. Holder nodes will always own the containers (and related applications), even if they are hosted on other nodes. When deploying a new container, the holder node sends a container deploy event through the DEL module. First, fog nodes send their reply and vote objects and the cluster selects the most suitable node to host the container (i.e., the solver node). When the holder node knows the identity of the chosen host node, the holder registers the container instance in the DCR publishing the required metadata to execute the container. These metadata are detailed in Table 3. Once the container instance is registered, the management smart contract notifies the host node, which gets the container metadata from the DCR and executes the container locally. To finish the deployment process, the host node solves the DEL event and labels the container as active in the DCR (both actions encapsulated in the same transaction).

Registering containers and their metadata in the DCR module allows the fog nodes to locate the applications executed in the cluster, since nodes are able to query the host IP address from its DDR smart contract and the network ports of each registered container. On the other hand, the DCR module allows to intrinsically establish the desired cluster state and the number of container replicas. The DCR maintains the containers that should be running in each cluster fog node, which allows a desynchronized node (for example, due to hardware or software failures, loss of power supply and so on) to recover its container desired state. Note that the DCR stores container instances, so it does not specify the number of container replicas as a single metadata. If multiple replicas of the same container are required, the holder node registers them separately (i.e., an applicant node sends multiple cluster events).

Other basic actions that influence the DCR module are migrating, updating and deleting (i.e., deactivating) containers. When a container is migrated, a new instance is created in the DCR. This allows to recover all the previously executed instances, increasing the cluster auditability. The container migration workflow is similar to the container deployment workflow shown in Fig. 5: (1) an applicant node sends a container migration event, (2) the cluster chooses a host or solver node, (3) the container holder registers a new instance in the

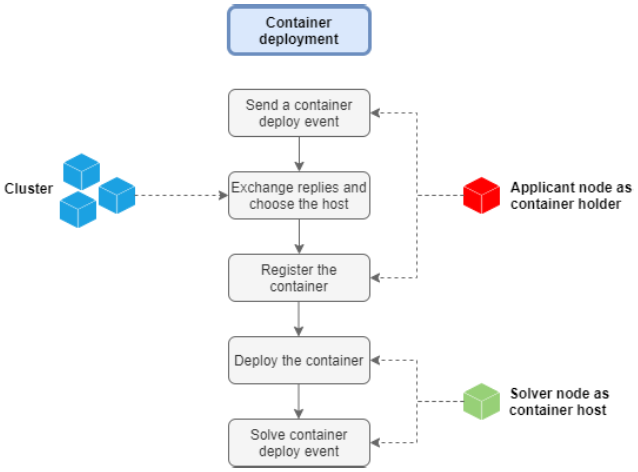


FIGURE 5. Container deployment workflow.

DCR and, (4) the host node executes the container and solves the event. When the host node solves the event and labels the container as active, the previous host node, i.e., the applicant node, is notified to perform the required post-migration tasks (resource release). Note that holder nodes are the only nodes which send container management events related to their own containers. These events are able to modify the state of the containers regardless of the host node that executes them.

The management smart contract shared between functional modules stores the list of active and inactive containers and implements the required procedures to manage container registrations. As in the DEL module, these procedures contain the log asynchronous triggers that notify fog nodes any state change in containers. The DCR module links a unique identifier to each registered container. This allows nodes to identify, in a distributed way, a container instance in the cluster. Therefore, it is not necessary to manage the long container identifiers that Docker generates by default (using SHA256 hash function) when creating a container. The management smart contract maintains its own container indexing system using a simple and anti-tampering integer counter.

4) DISTRIBUTED REPUTATION SYSTEM (DRS)

Once fog nodes are registered in the cluster through the DDR module, they are able to interact with the DEL and DCR modules through the management smart contract. Thus, fog nodes can send cluster events and deploy applications via containers. Although these actions are fundamental and all fog nodes will carry them out, a reputation system is necessary to limit the uncontrolled execution of such actions. The reputation system will reward nodes that follow the workflow rules and contribute more resources to the cluster. Moreover, it will penalize erratic decisions or bad behavior. The DRS module implements the reward and penalty mechanism that is used to incentivize cluster nodes. This mechanism manages the reputation value for each participating fog node. Reputation values quantitatively reflect the past behavior and the degree of trust and participation of cluster nodes. Additionally, they allow quantifying other features, such as availability or computational power of nodes. This is achieved by rewarding nodes with a longer meantime between failure (MTBF), nodes with self-healing properties that report less time to repair and nodes contributing a greater amount of resources to the cluster. Note that the reputation system focuses on evaluating the behavior of cluster devices that perform management and orchestration tasks, i.e., fog nodes. Measuring the behavior of IoT devices linked to fog nodes is out of the scope of this paper. Nevertheless, approaches proposed in the literature, such as the ones found in [48] based on network communications and collaboration between IoT devices, can be integrated into the DRS module to this end.

The DRS module implements the reputation system logic in a smart contract, different from the management smart contract, called *reputation smart contract*. This smart contract acts as an intermediary between the procedures implemented by the other functional modules and the nodes' reputation values. To increase the decentralization and transparency of the reputation mechanism, reputation values are stored exclusively in the isolated smart contracts deployed by the DDR module when new fog nodes are registered. Note that only the reputation smart contract can modify nodes' reputation values. Aside from containing the logic required to apply reputation variations (either positive or negative variations) to the cluster nodes, the reputation smart contract stores a public and immutable list containing the reputable actions available in the cluster. These actions correspond to the procedures implemented by the DEL and DCR modules in the management smart contract. Reputable actions always cause a variation in the reputation of the cluster node that executes them.

In order to prevent malicious execution of reputable actions and incentivize fog nodes to participate in the cluster, the reputation smart contract links two parameters to each reputable action, namely, the minimum reputation value required to execute the action and the reputation variation. Thus, cluster nodes always know the reputation required to execute each action and the reputation profit or reputation cost that it entails. Specifically, the reputable actions implemented by the management smart contract are related to sending DEL events (sending event, reply and vote objects and solving events) and managing container instances in the DCR (registration and deletion).

The DRS workflow is carried out when a node executes a reputable action in the cluster. Before executing the action, the management smart contract verifies that the fog node has enough reputation value to execute the reputable action. To this end, the management smart contract queries both the fog node smart contract and the reputation smart contract. If the reputation requirement is met, the reputable procedure executes the tasks associated with the reputable action and the reputation variation is applied to the cluster node.

C. CLIENT DESIGN SPECIFICATIONS

This section describes the most remarkable design aspects of the software client. As detailed above, the software client

divides its logic and basic actions into two modules: the Monitor and the Manager. The Monitor focuses on identifying outliers in resource utilization according to the pre-configured rules and the Manager receives this monitoring information to enforce the container orchestration by executing the relevant tasks. The software client manages and enforces all system basic actions following the design aspects described below:

- **Ethereum addresses management**. Since the DDR module registers fog nodes and IoT devices, the client must manage the files containing the cryptographic keys corresponding to the Ethereum accounts of those devices. To do this, each fog node maintains an address wallet. Note that the wallet manages the accounts used to interact with the system's Ethereum blockchain service. Moreover, the client software also stores the addresses of the smart contracts of the different functional modules. These addresses are needed to send transactions and execute the smart contracts.
- **Ethereum transactor**. The wallet not only manages the Ethereum accounts, but also prepares the transactions to be sent by the Manager module to the other cluster nodes. Note that the client wallet needs to manage and configure different transaction parameters such as the account nonce (the number of the next transaction to be sent by that account), the gas price and the gas limit for each smart contract storage function. More information about Ethereum transactions can be found in [49].
- **Monitoring system**. In the process of monitoring fog node resources, the client specifies a monitoring interval T in milliseconds which sets how often the dynamic metadata are collected and the predefined rules are checked. Other parameters of interest are R and S. R refers to the number of rules to be checked every monitoring interval, and S is the number of previous monitoring states stored to avoid usage peaks. Through parameter S, the client software can check, on the one hand, if during the previous monitoring states the same problem has been detected (avoiding usage peaks), and, on the other hand, if the fog node is already handling a similar problem, which avoids sending duplicate alerts from the Monitor to the Manager and ultimately, to the cluster.
- **Rule configuration**. Cluster nodes configure the rules according to their resource specification constraints and the cluster requirements (which will change over time). At each monitoring interval, the dynamic metadata obtained by the Monitor module are compared with the rules to discover anomalous usage values. Rules are composed of a data structure containing the parameters required to check and enforce them. Cluster nodes preconfigure their rules by setting up parameters such as the type of resource, the type of comparison that is carried out (arithmetic comparison, string comparison and so on), a specific value or a threshold to be enforced and the action to be taken by the Monitor module if the rule is thrown. Note that fog nodes specify the amount of resources they provide to the cluster by setting a default rule for each type of resource that the system monitors. After that, cluster nodes can define additional rules using different types of verbosity, thresholds and actions to be performed by the Monitor module.
- **Broadcast selection algorithms**. Mainly, two heuristic algorithms are applied to solve a broadcasted event in the cluster: the solver selection algorithm and the container selection algorithm. The first one deals with the selection of the most suitable fog node to solve an issue in the cluster. For this, the static and dynamic metadata of the N fog nodes are taken into account. This algorithm selects a solver node depending on several factors, such as the node with the highest available resource capacity, or the geolocation of cluster devices. The second algorithm selects locally the most suitable running container to be migrated to another fog node, releasing resources in the applicant node. The selection takes into account the maximum amount of allocated resources and the current resource usage of each container that the applicant fog node runs. Additionally, other container metadata is taken into account, such as the container impact in the cluster, the type of application it hosts and the main resource used by such application (CPU, memory, disk or I/O). For example, for a container hosting a database, the main resource is the disk storage capacity, and regardless of the other container metadata, if a storage issue occurs on the node, the container will be more likely to be migrated. Note that different techniques can be applied on the broadcast selection algorithms being a tradeoff between cost and efficiency. The algorithms described in this paper are an efficient solution to obtain a first prototype.
- **Asynchronous log handling**. The client manager listens to all asynchronous logs that are produced in the smart contracts. In this way, fog nodes are able to perform specific tasks in response to actions performed by other cluster nodes, i.e., in response to the execution of smart contracts by other nodes. In addition, the log system also allows asynchronously notifying the fog nodes when cluster state changes occur (e.g., to notify about a container metadata modification).

VI. EVALUATION AND RESULTS

Following the proposed architecture, the design of the different functional modules and the orchestration techniques described above, a simplified HIDRA prototype has been implemented in order to demonstrate the functionality of the proposal. Also, some preliminary performance results are provided which illustrate the feasibility of the proposed architecture even on fog nodes with constrained resources. Thus, in Section VI-A the hardware and software characteristics of the prototype are detailed, while Section VI-B evaluates the feasibility of the proposal in terms of functionality, resource usage and power consumption.

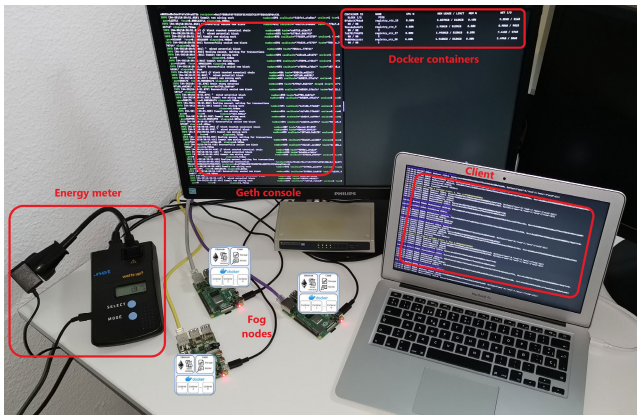


FIGURE 6. HIDRA testbed.

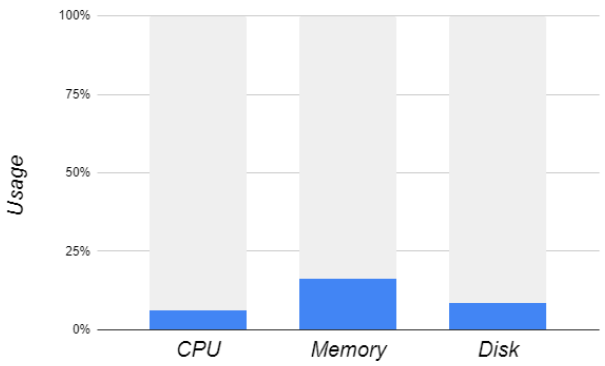


FIGURE 7. Prototype resource usage.

A. PROTOTYPE SPECIFICATIONS

HIDRA is independent of the type of resources that implement the fog layer. One common and sustainable possible implementation of fog nodes consists of the use of single-board computers (SBCs). In order to check the suitability of the proposal for diverse fog environments, an implementation on a popular type of SBCs, namely, the Raspberry Pi, has been developed. Fig. 6 shows the deployed HIDRA testbed at the Albacete Research Institute of Informatics (I3A). Each one of the fog nodes implements the internal details depicted in Fig. 1. The picture shows multiple terminals related to one of the fog nodes. Specifically, the terminals show the Geth console monitoring the blockchain, the running containers and the client logs. Note that the laptop is only used for monitoring purposes. Additionally, the picture shows the Watts Up .NET power meter, which has been used to obtain power consumption metrics, as it will be detailed later.

The prototype implements the main workflow protocol and all the functional modules of the proposal to carry out resource management and orchestration. The prototype consists of a cluster formed by $N = 3$ equal fog nodes, even though the proposal allows connecting a heterogeneous set of devices with different specifications. Each fog node is an SBC device, specifically, Raspberry Pi 4 Model B with 1.5GHz quad core CPU, 4GB of memory and 64GB of disk storage. Regarding the software, each Raspberry runs the Raspbian Buster Lite operating system and the key components described in the Section IV-B. Thus, each fog node runs a blockchain service, a software client that monitors and interconnects with other modules and a light virtualization service.

The Ethereum platform is used to implement the blockchain service component. Each fog node in the cluster acts as a full blockchain service storing the entire blockchain. This allows fog nodes to locally maintain the cluster state and smart contracts to execute them when needed. In particular, the blockchain service running on each node deploys Geth 1.9.16, an implementation of Ethereum written in the Golang programming language. To start a private blockchain in the cluster, a genesis block is initially generated. This block specifies different configuration parameters such as the default sealer nodes, the network ID or the block generation period (time between block sealing).

The software client has been implemented in the Go programming language, which offers great compatibility with Geth when interacting with the blockchain service. On the other hand, smart contracts are implemented in Solidity, an object-oriented and high-level programming language designed to target the EVM [50]. Note that the asynchronous logs described in previous sections refer to the Solidity events emitted by smart contracts in response to an inbound transaction. Such events are stored in the blockchain as logs and cannot be accessed from smart contracts. The client continuously listens to the different Solidity event types. When an event is received, the client triggers the required management tasks related to the received event (for example, the sending of a reply object in response to an event object).

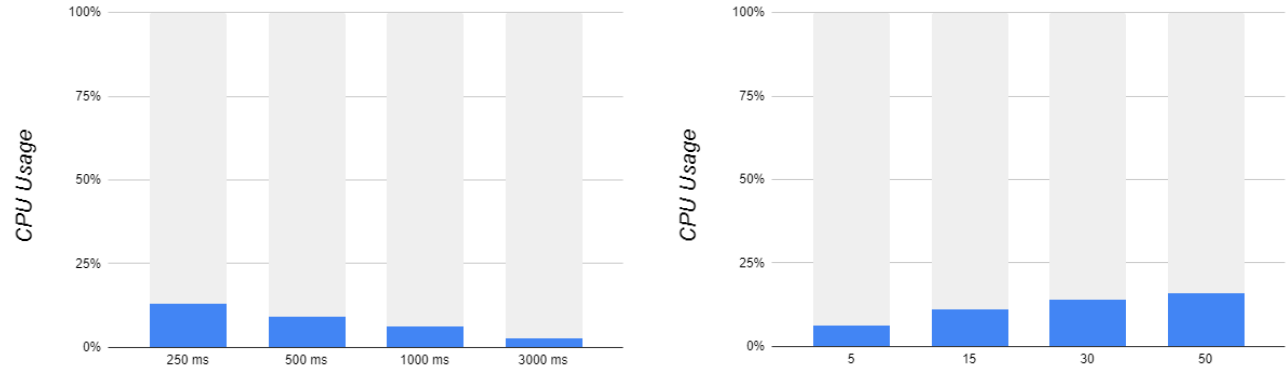
Lastly, Docker 19.03.12 is used as the light virtualization service. Using the Docker Engine HTTP API and the Docker Engine Go SDK, the client is able to implement the functionality required to manage all the resources of the docker daemon (containers, images, volumes and so on).

B. PROPOSAL VALIDATION

In this section, some experiments aimed at validating the implementation of HIDRA are presented. More specifically, the basic resource usage and power consumption of the system has been measured to evaluate the overhead introduced in the fog nodes. The most demanding use case (i.e., a DEL container migration event) has also been considered, in order to assess the feasibility of the proposal in a worst case scenario.

1) PROTOTYPE RESOURCE COSTS

First we will analyze the resource usage when the system is running in stable mode, meaning that the fog nodes are running the required services (i.e., the Geth service, the management client and the Docker daemon) and no notification has been generated. We will generically call this state the *monitoring mode*. To be more precise, in the monitoring mode the Monitor client obtains the node dynamic metadata and checks the configured rules every T time interval with no need to change the cluster state, so no Ethereum transactions



(a) T parameter.

(b) R parameter.

FIGURE 8. CPU usage comparison.

are generated or sent. The other parameters that are taken into account in the monitoring mode are the number of rules to be checked (R) and the number of previous monitoring states (S). Fig. 7 shows the CPU, memory and disk storage usage on a fog node working in the monitoring mode. These results have been obtained with $T = 1000$, $S = 10$ and $R = 5$ as default values. The obtained results show a low resource utilization by the Control Plane (less than 7% CPU, 20% memory and 10% disk), so most of the fog node resources can be allocated to the execution of payload, in our case, services deployed as Docker containers.

2) MONITORING COMPUTATION COSTS

Fog nodes support different configuration parameters in the monitoring mode. Fig. 8 shows the computational cost incurred in a node depending on the values of the configuration parameters T and R. Results show that the smaller the monitoring interval T and the higher number of checked rules per measure, the higher CPU usage in the fog node. Although a linear relationship between CPU usage and parameters T and R can be observed, results show the feasibility of the proposal even with unusual parameterization values. Maximum CPU usage is under 20% in all the cases. Note that, depending on the fog environment and the use case, different parameterization may be needed that condition the operation of the Monitor module. Even within the same use case, the parameters of each fog node could be configured differently according to its needs. It is worth noting that the S parameter does not influence the resource utilization of the fog node, although it does influence the time required to discover anomalous utilization values in the node.

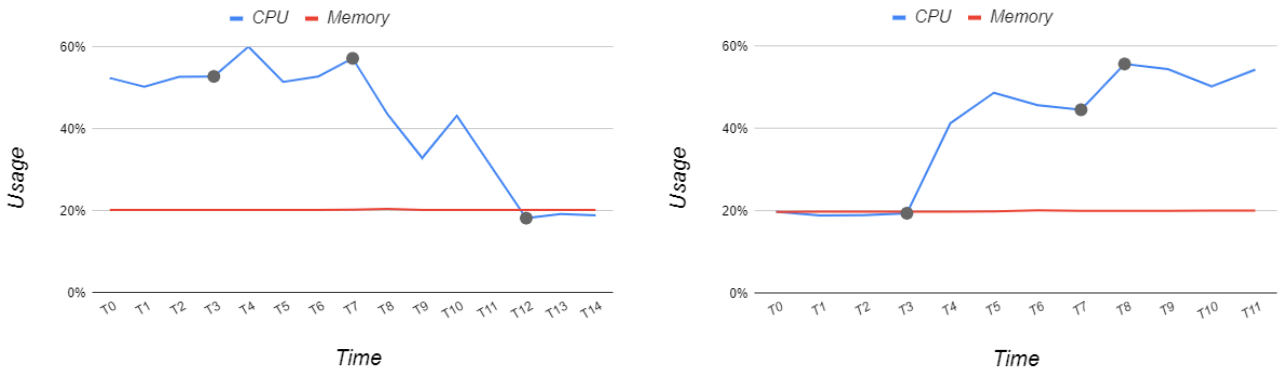
3) DEL RESOURCE COSTS

In addition to the monitoring task, fog nodes generate events such as container migration or deletion. These events imply updating the cluster state, which impacts the performance and cost of the proposal. Thus, this section analyzes the cost in terms of resource usage during a DEL container migration event, the event that consumes the most node and network resources due to its complexity (see Section V-A for more details).

Fig. 9(a) and Fig. 9(b) show the CPU and memory usage during a container migration event on the applicant and solver fog nodes, respectively. In particular, the experiment migrates a CPU demanding application implemented as a Nginx-based web server container that receives requests at a constant rate. It must be noted that the Docker image used by the container running on the applicant node is already available in the solver node (i.e., fog nodes have preloaded images, so that the applications can be quickly spinned up when needed). Results show a constant memory usage during the migration, but the CPU usage varies.

In the applicant fog node (see Fig. 9(a)), three key time instants can be observed related to the CPU usage: a) time $T3$, where the container migration event is generated due to the fact that the CPU limit set in the pre-configured rules is exceeded; b) time $T7$, when the DEL workflow is completed (step 7 of Fig. 3); and c) time $T12$, when the node returns to its monitoring mode after executing the ending tasks (stop the problematic container and delete it from the DCR). From $T12$, resources are released in the applicant fog node and the CPU usage is due to the monitoring mode and other applications. Note that to improve the availability of the applications running in the cluster, the applicant node does not delete the container instance until the event is solved and the solver node executes a new container instance.

Accordingly, Fig. 9(b) shows the CPU used by the solver node during the migration of the container. Again, there are a few key time instants that must be highlighted: a) the new container starts its deployment in the solver node in time $T3$ (when the exchange of DEL objects has already done), which increases the CPU usage in the node; b) the deployment concludes in time $T7$ so hereinafter the container is ready, the cluster event is solved and the applicant node is notified; and c) in time $T8$, the migrated container attends the web requests and the CPU usage increases up to 56%, a CPU usage similar to which the applicant node had in the first time instants. It is worth noting that the CPU and memory usage observed from $T3$ onward can also correspond to a container



(a) Applicant node.

(b) Solver node.

FIGURE 9. Resource usage during a DEL migration.

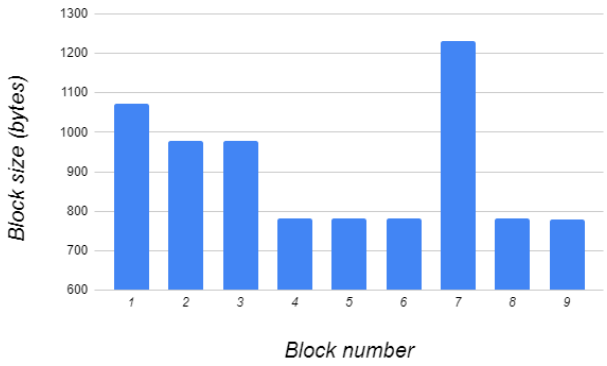


FIGURE 10. Blocks during a DEL migration.

deploy event, that is, a container that is started as a response to a client request instead of as a result from a migration event.

Just to further illustrate the behavior of the prototype, some insight into latencies has been measured. The total latency of the DEL workflow during the container migration event was about 1.95 seconds. More specifically, the deployment of the container on the solver node took most of the process, about 1.66 sec. This means that the control management of the deployed prototype took less than 0.3 sec. Note that the latency of the process may vary depending on the type of container being executed and its configuration.

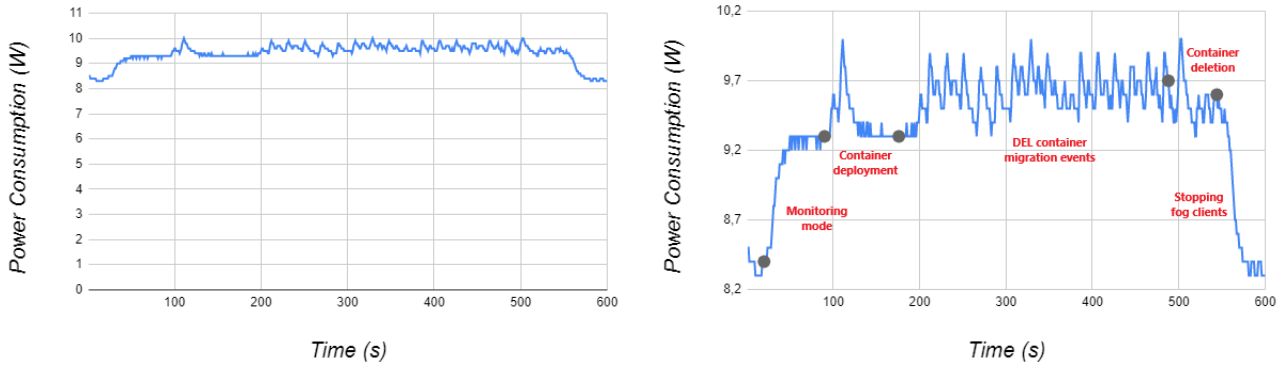
Finally, in order to evaluate the utilization of blockchain service resources during a DEL container migration event, the disk storage used by each block generated in the process is measured. Fig. 10 shows the size in bytes of each block. Considering that our prototype consists of three fog nodes, nine blocks of different sizes are generated. Note that each of the blocks contains a single transaction due to the instant block generation period configured in the blockchain service. More precisely, the first block corresponds to the event object, including the applicant reply object. The next two blocks are the remaining reply objects. Blocks four, five and six correspond to the vote objects. The next block refers to the new container instance registration in the DCR. Block eight solves the cluster event and, finally, the last block corresponds to the deletion of the problematic container instance from the DCR. In total, each DEL workflow generates about 8 KB of disk storage. This value depends on the N parameter and the size of the metadata sent in the process. Thus, for a cluster with N nodes, $2N + 3$ blocks will be generated and the storage used by the blockchain service will increase linearly. Results show a non-data intensive storage demand. Nevertheless, the storage resources in the fog nodes are scarce, so sharding techniques [51] or hierarchical approaches [52] could be applied to reduce the storage demand of the system.

4) PROTOTYPE POWER CONSUMPTION

It is well known that power consumption is a critical drawback of blockchain technologies, due mainly to its mining process. HIDRA uses a permissioned blockchain and a PoA consensus algorithm in order to keep computational load into reasonable levels, as shown in the previous section. In this section, the power consumption of the testbed where the HIDRA prototype has been deployed is monitored and analyzed. To this end, the power meter Watts Up .NET has been used. This meter provides an independently managed and accessed power data collection mechanism. In all our experiments, the monitoring frequency has been set up to one sample per second (the maximum allowed by the meter).

The information obtained using Watts Up .NET reflects the power consumption of the prototype fog nodes ($N = 3$). The experiment has consisted of measuring power consumption for 10 minutes, where each fog node initially deploys three Nginx-based web server containers. During the experiment, the deployed containers are migrated between the different cluster nodes using DEL container migration events. The monitoring parameter configuration used in the experiment has been the same as that used in Fig. 7 to measure the prototype resource costs ($T = 1000$, $S = 10$ and $R = 5$).

Fig. 11(a) exposes the testbed power consumption during the experiment. It is observed how the power consumption varies between a minimum of 8.3 watts per second and 10 watts per second throughout the experiment, i.e., there is at most an increase of 20.5% compared to the minimum power consumption. On the other hand, Fig. 11(b) shows a zoom-in on the upper part of the Fig. 11(a)



(a) Overall view.

(b) Key instants and intervals.

FIGURE 11. Testbed power consumption.

vertical axis, so that some key instants and intervals can be more clearly seen. The interval labeled as *monitoring mode* refers to the period in which all the fog nodes initialize their clients. When this period ends, the fog nodes are running the blockchain service, the light virtualization service and the software client monitoring their own resources, but they do not deploy containers or send transactions. At the beginning of the *container deployment* interval, three Nginx containers are deployed on each node. In this interval, an increase in power consumption is observed during the deployment process, but the power consumption remains stable after the containers have been deployed. During the *DEL container migrations events* interval, the cluster carries out container migrations, which involves continuously sending transactions and deploying/deleting containers. To end the experiment, the containers running on each fog node are deleted and the clients stop running (*container deletion* interval and *stopping fog clients* interval respectively). Summarizing the results, the experiment has consumed an average of 9.39 watts per second carrying out 45 DEL container migration events.

VII. DISCUSSION

This section summarizes the main features of the distributed architecture HIDRA proposed in this paper. The deployment of a cluster of fog nodes interconnected through the Ethereum blockchain service, the implementation of the software client and the use of light virtualization technologies has allowed us to verify the suitability of the proposal. The properties of the proposed architecture for resource management are manyfold: (1) a choreography approach is designed to manage the fog nodes of the cluster. The master-client architecture used by well-known orchestration architectures such as Kubernetes, Mesos or Docker Swarm has been avoided. The control plane of the cluster is not centralized on a master node to avoid single points of failure. Moreover, the distributed control among all nodes in the cluster increases the security and defends the system against denial of service (DoS) attacks and physical attacks. (2) an event-driven architecture with migration and replication protocols for containerized applications is designed. The implementation of these protocols provides self-healing properties, as well as fault tolerance and high availability to HIDRA. (3) blockchain technology is used to cope up with the management of constrained fog nodes, and different smart contracts are implemented to support the infrastructure in the fog layer. These smart contracts are the engine to update and optimize the configuration of the system in a dynamic and autonomous manner. Moreover, the inherent properties of blockchain ensure data immutability preventing malicious nodes or third parties from manipulating them.

Concerning the blockchain service, it is worth pointing out the design decisions taken. First, the implementation of a permissioned blockchain versus a permissionless blockchain improves certain properties such as the scalability of the devices and the transaction throughput (since the number of participating fog nodes is limited) and increases the customizability of the network. Note that a permissioned blockchain offers a balance between transparency and privacy, since it allows storing the information transparently to the involved nodes while protecting the data from public third parties. The above features make permissioned blockchains suitable for a large number of use cases in Fog-IoT environments [24]. On the other hand, the blockchain service uses the PoA algorithm to establish consensus. This algorithm does not need to compute expensive cryptographic operations like the PoW algorithm, so the cluster saves resources and energy. The PoA algorithm is a good option to reach the consensus in environments where there is no trust between the network nodes. This enables use cases of private blockchains where the participants a priori do not know each other but work together to maintain the blockchain.

Through the Monitor module implemented in the client software, cluster fog nodes are able to monitor their resource usage every T time interval, which allows the usage values to be checked against the rules configured in each node. The rules allow fog nodes to both discover anomalous usage values and set the amount of resources that a node provides to the cluster. If the cluster reaches its resource usage limit, it is able to dynamically increase the limits set by fog nodes (as long as the physical resource specifications are not exceeded). On the other hand, the Manager module manages the event flow of the main workflow protocol in

each fog node by implementing interfaces between the different functional modules. The management and orchestration decisions made by each individual fog node are the same for all of them due to the same Manager module running in each node. The choice of solver nodes is made in a consensual way, which avoids fraudulent and individual decisions generated by nodes acting outside the established protocol, thus avoiding unfair or unbalanced resource management and orchestration.

The proposed functional modules work together in the management and orchestration of applications. Each module provides the system with certain features and benefits. It is worth noting some features of the DRS module. The DRS aims to avoid event flooding attacks in the cluster, which can degrade the system performance (for example, by deploying a large number of duplicate containers). Note that only registered fog nodes with enough reputation value are able to change the cluster state. In order to prevent Sybil attacks, the DDR module can be supervised off-chain by the local administrative entity that manages the system, i.e., controlling the devices that are registered. Another option to hinder Sybil attacks could be to assign low initial reputation values to the new fog nodes that are registered. Thus, nodes need an initialization time interval in the cluster until they can use the cluster resources. In this time interval, fog nodes simply offer their resources to the cluster and participate in the events of other nodes. It depends on the administrative entity to choose the most appropriate reputation settings for each use case (reputation limits and variations of each reputable action). Regarding the DEL and DCR modules, the DEL workflow interacts with the DCR to manage cluster container instances. The DCR is accessible by all cluster nodes, enabling them to know the metadata of any container at any time, which increases the transparency of the container registration process. This also allows HIDRA to increase the resilience and availability of the cluster applications, since if a fog node cannot recover from a failure, other cluster nodes will query the DCR and deploy the erratic node's containers.

Finally, a proof-of-concept implementation of HIDRA has been deployed over a prototype based on SBC nodes. Results show the suitability of the proposal regarding the low resource usage and power consumption of the Control Plane, even in fog computing environments with limited capacity devices. Moreover, the chosen technologies for the implementation of the system allow to increase its heterogeneity. So, it is possible to set up a cluster of fog nodes with different architectures, operating systems and resource specifications that is very usual in current fog and IoT environments.

VIII. CONCLUSION

In this paper, we proposed HIDRA, a decentralized fog architecture for Fog-IoT environments leveraging the adoption of both permissioned blockchain networks and lightweight container-based virtualization technologies. This distributed fog architecture has been designed to be fault-tolerant, secure and auditable. Moreover, it has been designed to facilitate future extensions and its implementation is based on open source technologies. So, a proof-of-concept prototype based on a testbed of Raspberry Pi nodes has been built to verify the feasibility of the proposal. The preliminary evaluation has revealed some promising performance results even on constrained nodes.

For future work, we will study the scalability of the proposal to execute reliable container-based applications for IoT. Furthermore, it is a must to use an optimal distributed resource management in order to improve the cost-energy of the system. Hence, some machine learning techniques will be adopted to allocate the containers in the system. Another direction is to explore other reputation techniques to ensure trusted fog nodes.

REFERENCES

[1] E. Borgia, "The Internet of Things vision: Key features, applications and open issues," *Comput. Commun.*, vol. 54, pp. 1–31, Dec. 2014.
[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Workshop Mobile Cloud Comput. (MCC)*, New York, NY, USA: Association for Computing Machinery, 2012, pp. 13–16.
[3] *Cisco Devnet. What is IOx*. Accessed: Feb. 25, 2021. [Online]. Available: https://developer.cisco.com/docs/iox/
[4] L. Dignan. *What's Next for Data Centers? THink Micro Data Centers*. Accessed: Feb. 25, 2021. [Online]. Available: https://www.zdnet.com/article/whats-next-for-data-centers-think-micro-d%ata-centers/
[5] Raspberry Pi Foundation. *Raspberry PI Official Webpage*. Accessed: Feb. 25, 2021. [Online]. Available: https://www.raspberrypi.org
[6] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia, "Fog computing: A comprehensive architectural survey," *IEEE Access*, vol. 8, pp. 69105–69133, 2020.
[7] J. Carrillo-Mondejar, J. M. Castelo Gomez, C. Núñez-Gómez, J. R. Gómez, and J. L. Martínez, "Automatic analysis architecture of IoT malware samples," *Secur. Commun. Netw.*, vol. 2020, pp. 1–12, Oct. 2020.
[8] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.
[9] The Kubernetes Authors. *Kubernetes: Production-Grade Container Orchestration*. Accessed: Feb. 25, 2021. [Online]. Available: https://kubernetes.io/
[10] KubeEdge Project Authors. *KubeEdge: An Open Platform to Enable EDge COmputing*. Accessed: Feb. 25, 2021. [Online]. Available: https://kubeedge.io/en/
[11] K3 Project Authors. *K3s: Lightweight Kubernetes*. Accessed: Feb. 25, 2021. [Online]. Available: https://k3s.io/
[12] C. Avasalcai, I. Murturi, and S. Dustdar, *Edge Fog: A Survey, Use Cases, and Future Challenges*. Hoboken, NJ, USA: Wiley, 2020, ch. 2, pp. 43–65.
[13] A. A. Sadri, A. M. Rahmani, M. Saberikamarposhti, and M. Hosseinzadeh, "Fog data management: A vision, challenges, and future directions," *J. Netw. Comput. Appl.*, vol. 174, Jan. 2021, Art. no. 102882.
[14] M. Di Pierro, "What Is the blockchain," *Comput. Sci. Eng.*, vol. 19, no. 5, pp. 92–95, 2017.
[15] N. Szabo, "Smart contracts: Building blocks for digital markets," *Extropy, J. Transhumanist Thought*, vol. 16, no. 18, pp. 1–10, 1996.
[16] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-enabled smart contracts: Architecture, applications, and future trends," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 11, pp. 2266–2277, Nov. 2019.
[17] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Application management in fog computing environments: A taxonomy, review and future directions," *ACM Comput. Surveys*, vol. 53, no. 4, pp. 1–43, Sep. 2020.
[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[19] O. Salman, I. Elhajj, A. Chehab, and A. Kayssi, "IoT survey: An SDN and fog computing perspective," *Comput. Netw.*, vol. 143, pp. 221–246, Oct. 2018.

[20] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, Nov. 2018.
[21] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–37, Oct. 2019.
[22] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Gener. Comput. Syst.*, vol. 87, pp. 278–289, Oct. 2018.
[23] M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and A. Colman, "Blockchain consensus algorithms: A survey," 2020, *arXiv:2001.07091*. [Online]. Available: https://arxiv.org/abs/2001.07091
[24] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1676–1717, 2nd Quart., 2019.
[25] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain," in *Proc. Italian Conf. Cyber Secur.*, Jan. 2018, pp. 1–11.
[26] V. Buterin, "A next-generation smart contract and decentralized application platform," Tech. Rep., 2013. [Online]. Available: https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf
[27] *Geth*. Accessed: Feb. 25, 2021. [Online]. Available: https://geth.ethereum.org/
[28] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource management approaches in fog computing: A comprehensive review," *J. Grid Comput.*, vol. 18, no. 1, pp. 1–42, Mar. 2020.
[29] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for Internet of Things services," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 16–24, Mar. 2017.
[30] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Resource provisioning for IoT application services in smart cities," in *Proc. 13th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2017, pp. 1–9.
[31] K. Velasquez, D. P. Abreu, M. R. M. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, E. Monteiro, and E. Madeira, "Fog orchestration for the Internet of everything: State-of-the-art and research challenges," *J. Internet Services Appl.*, vol. 9, no. 1, pp. 1–23, Dec. 2018.
[32] K. Velasquez, D. P. Abreu, D. Goncalves, L. Bittencourt, M. Curado, E. Monteiro, and E. Madeira, "Service orchestration in fog environments," in *Proc. IEEE 5th Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2017, pp. 329–336.
[33] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2019, pp. 351–359.
[34] H. Baniata and A. Kertesz, "A survey on blockchain-fog integration approaches," *IEEE Access*, vol. 8, pp. 102657–102668, 2020.
[35] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.
[36] P. Singh, A. Nayyar, A. Kaur, and U. Ghosh, "Blockchain and fog based architecture for Internet of everything in smart cities," *Future Internet*, vol. 12, no. 4, p. 61, Mar. 2020.
[37] P. Kochovski, V. Stankovski, S. Gec, F. Faticanti, M. Savi, D. Siracusa, and S. Kum, "Smart contracts for service-level agreements in edge-to-cloud computing," *J. Grid Comput.*, vol. 18, no. 4, pp. 673–690, Dec. 2020.
[38] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, "FogBus: A blockchain-based lightweight framework for edge and fog computing," *J. Syst. Softw.*, vol. 154, pp. 22–36, Aug. 2019.
[39] Y. Jiao, P. Wang, D. Niyato, and K. Suankaewmanee, "Auction mechanisms in cloud/fog computing resource allocation for public blockchain networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 1975–1989, Sep. 2019.
[40] H. Wang, L. Wang, Z. Zhou, X. Tao, G. Pau, and F. Arena, "Blockchain-based resource allocation model in fog computing," *Appl. Sci.*, vol. 9, no. 24, p. 5538, Dec. 2019.
[41] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, and Y. Zhao, "EdgeChain: An edge-IoT framework and prototype based on blockchain and smart contracts," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4719–4732, Jun. 2019.
[42] R. A. Memon, J. P. Li, M. I. Nazeer, A. N. Khan, and J. Ahmed, "DualFog-IoT: Additional fog layer for solving blockchain integration problem in Internet of Things," *IEEE Access*, vol. 7, pp. 169073–169093, 2019.
[43] T. Kumar, E. Harjula, M. Ejaz, A. Manzoor, P. Porambage, I. Ahmad, M. Liyanage, A. Braeken, and M. Ylianttila, "BlockEdge: Blockchain-edge framework for industrial IoT networks," *IEEE Access*, vol. 8, pp. 154166–154185, 2020.
[44] H. L. Cech, M. Grossmann, and U. R. Krieger, "A fog computing architecture to share sensor data by means of blockchain functionality," in *Proc. IEEE Int. Conf. Fog Comput. (ICFC)*, Jun. 2019, pp. 31–40.
[45] *Docker*. Accessed: Jan. 27, 2021. [Online]. Available: https://www.docker.com/
[46] G. Peralta, M. Iglesias-Urkia, M. Barcelo, R. Gomez, A. Moran, and J. Bilbao, "Fog computing based efficient IoT scheme for the industry 4.0," in *Proc. IEEE Int. Workshop Electron., Control, Meas., Signals Their Appl. Mechatronics (ECMSM)*, May 2017, pp. 1–6.
[47] B. Mishra and A. Kertesz, "The use of MQTT in M2M and IoT systems: A survey," *IEEE Access*, vol. 8, pp. 201071–201086, 2020.
[48] D. Nabil, D. Tandjaoui, I. Romdhani, and F. Medjek, "Trust management in Internet of Things," in *Security and Privacy in Smart Sensor Networks*, 1st ed. IGI Global, May 2018, pp. 122–146, doi: 10.4018/978-1-5225-5736-4.ch007.
[49] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Tech. Rep., 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf
[50] *Solidity*. Accessed: Jan. 28, 2021. [Online]. Available: https://solidity-es.readthedocs.io/es/latest/
[51] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proc. Int. Conf. Manage. Data*, Jun. 2019, pp. 123–140.
[52] G. Wang, Z. Shi, M. Nixon, and S. Han, "ChainSplitter: Towards blockchain-based industrial IoT architecture for supporting hierarchical storage," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 166–175.

CARLOS NÚÑEZ-GÓMEZ graduated in computer engineering from the University of Castilla-La Mancha, in 2016. He received the M.Sc. degree in information and communication security from the Universitat Oberta de Catalunya, in 2019. He is currently pursuing the Ph.D. degree in advanced computer technologies with UCLM. In 2016, he started working as a Software Developer and a Project Manager in the industrial sector. He is also a member of the High-Performance Networks and Architectures Group (RAAP), Albacete Research Institute of Informatics (I3A). His research interests include blockchain, information security, and fog computing environments.

BLANCA CAMINERO (Member, IEEE) received the Ph.D. degree in computer science from UCLM. She is currently an Associate Professor in computer architecture and technology with the Department of Computing Systems, UCLM. She has been teaching networking related subjects at the School of Computer Science and Engineering, Albacete, since 2000. She also carries out her research at the High-Performance Networks and Architectures Group (RAAP), Albacete Research Institute of Informatics (I3A). Her current research interests include QoS support and efficient resource scheduling in distributed systems, such as cloud, fog, and edge.

CARMEN CARRIÓN (Member, IEEE) received the Ph.D. degree in physics from the University of Cantabria. She is currently an Associate Professor in computer architecture and technology with the Department of Computing Systems, University of Castilla-La Mancha. Her research interests include higher education, resource management schemes, virtualization technologies, and the QoS in fog-IoT frameworks.