Received 24 September 2022; revised 1 November 2022; accepted 8 November 2022. Date of publication 11 November 2022; date of current version 22 November 2022.

Digital Object Identifier 10.1109/OJCOMS.2022.3221667

Exploring the Intersection of Consortium Blockchain Technologies and Multi-Access Edge Computing: Chronicles of a Proof of Concept Demo

ANGELO VERA-RIVERA[1] (Student Member, IEEE), EKRAM HOSSAIN[1] (Fellow, IEEE), AND AHMED REFAEY HUSSEIN[2] (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB R3T 2N2, Canada

[2]School of Engineering and Physical Sciences, University of Guelph, Guelph, ON N1G 2W1, Canada

CORRESPONDING AUTHOR: E. HOSSAIN (e-mail: ekram.hossain@umanitoba.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

ABSTRACT The efficient administration of network resources in Multi-Access Edge Computing (MEC) is an active research topic these days. Task sharing, in particular, is one of the fundamental problems regarding MEC architectures although the existing literature approaches the topic mostly from the mobile user point of view. In this article, we present the chronicles of *EdgeChain* network, a proof of concept demo of a blockchain-based model for secure and private task sharing collaboration specially designed for edge computing servers. *EdgeChain* operates on a decentralized approach that counts on the blockchain services provided by the Hyperledger Fabric platform. The network offers enhanced security features that leverage the permissioned nature of Fabric, more specifically, it relies on Fabric's membership service to validate identities and allowed behavior of participant nodes in the network. This design choice restrains external attackers from interfering with the normal operation of the task sharing scheme. The network also offers enhanced privacy features powered by a smart contract design that makes use of multiple decoupled Fabric channels with separate operation rules, ledgers, and peer-to-peer communication networks. This design choice guarantees that the computational tasks circulating in the network are only exposed to servers participating in task sharing services. The trait prevents other servers in the network to have access to private tasks and their data. The article goes through the multiple stages of the design and construction process of the proof of concept demo, from the blockchain-based task sharing framework and system model, all the way to the implementation details of the network.

INDEX TERMS Multi-access edge computing, next-generation communication networks, consortium blockchain, hyperledger fabric, task sharing.

I. INTRODUCTION

THE DEPLOYMENT of 5G networks will require a huge amount of financial investment from telecommunication companies. Network operators must find new revenue streams to level off the capital expenditure on infrastructure and technical support. Fortunately, the adoption of the 5G standard will also open the door to rethinking the ownership model for network operators that, in our opinion, is outdated and not well equipped to handle emerging business cases in the IT sector. Given the intense capital expenditure, infrastructure cooperation seems to be an alternative to lower infrastructure costs without affecting the proper delivery of applications and services for businesses and their consumers. In traditional ownership frameworks, telecommunication companies may reach rigid agreements in a centralized style where senior management teams get together and discuss static cooperation mechanisms that, in many cases, may include infrastructure sharing in any form or shape. Unfortunately, traditional frameworks for infrastructure ownership may not adapt very well to the needs

of emerging 5G business models that might require more dynamic and self-organizing approaches [1], [2], [3].

The decentralized services offered by blockchain technologies could become the facilitator for automated resource management schemes that adapt better to the complexities of sophisticated 5G applications. Blockchain networks provide a trust model for (1) record-keeping the truth of a network (i.e., tamper-proof distributed ledgers), and (2) secure interactions for network participants (i.e., immutable smart contracts). Unlike centralized networks in which trust comes directly from the reputation of a central node, the trust model in blockchain networks relies on distributed consensus algorithms and advanced cryptography to maintain truthfulness in the system. Smart contracts, on the other hand, are pieces of immutable software installed on the blockchain. They hold definitions, rules, data, and processes agreed by network participants to generate transactions that update the ledger. The self-execute nature of smart contracts allows network operators to automatically enforce cooperation agreements previously negotiated by participant organizations. Blockchain technologies might represent the opportunity to empower companies with a surfeit of resources to participate in automatic on-demand trading schemes that may otherwise be done statically.

The Hyperledger Fabric platform is a permissioned blockchain that offers membership, ledger, consensus, and chaincode services that support the implementation of multi-purpose distributed solutions. Fabric is a lightweight, low latency, and high throughput blockchain that may open opportunities for the implementation of distributed trading markets for network operators in the telecommunications sector. In blockchain-based marketplaces, network operators can trade scarce resources under competitive conditions in self-governed and secure platforms. From a market perspective, network resources might become scarce either because the resource supply is short as in the case of the frequency spectrum, or because production costs are high as in the case of computer processors. Moreover, the marketization of network resources in the telecommunications sector may impact network services such as wireless spectrum allocation, network slicing, and the lease of computer resources. Moving from static methods to on-demand smart contracts for infrastructure sharing may require blockchain-based market models that can summon multiple market players to trade network resources in both seller (provider) and buyer (requester) roles.

A Hyperledger Fabric-based multilateral marketplace would consist of a set of network operators willing to participate in the system and a set of smart contracts that facilitates the negotiation and trading of network resources. The Hyperledger Fabric platform brings in two key features to support the proper operation of the marketplace. First, it maintains a distributed record of all transactions in the network, and second, it supports the distributed execution of on-demand cooperation agreements through the use of immutable smart contracts. The low latency and high throughput characteristics of Fabric would make the execution of smart contracts in the marketplace highly dynamic and ductile. In a Fabric-based marketplace, market applications would send trading proposals to the blockchain network. The proposals contain requests for resources whose details depend on the implemented market model (i.e., regular auctions, reverse auctions, or any other form of resource allocation scheme). When trading transactions are received by the blockchain, the marketplace logic is automatically activated through the means of installed smart contracts. The terms of trading schemes are pre-negotiated by the market players and are subsequently embedded into smart contracts available in the market network. Trading models might be based on traditional resource allocation approaches that maximize global objectives defined by the players (e.g., multi-processor task scheduling problems), or game theory approaches where individually rational players seek to maximize their utility functions (e.g., auction problems). If trading proposals are successful, market players will endorse the transactions so that they become ready to be added to the blockchain network. Moreover, the ordering service will pack successful transactions into blocks that are later appended to the distributed ledger of the network. Once appended, trading transactions become part of the official truth of the market. This trading framework guarantees that market transactions are transparent and immutable. Also, it might provide an effective trust model for market players to deal with the contentious nature of the relationships among participants in the marketplace.

Although resource sharing is not a new idea and has been used extensively in previous generations of communication networks, sharing models typically consisted of long-term static agreements where operators accept to share network infrastructure without considering the short-term dynamic variations of resource demands. Also, the aggressive competition among market players in the telecommunications sector makes it almost impossible to have a trusted central intermediary to deal with tensions among market competitors. This is true especially when it comes to decisions on pricing and allocation of resources. On the other hand, the idea of decentralized marketplaces for infrastructure sharing has the potential to relieve market tensions, however, it requires the active participation of multiple actors in the industry including vendors, network operators, and regulators. In summary, blockchain platforms, Hyperledger Fabric, in particular, may power decentralized, transparent, and secure ownership models for emerging business applications based on 5G networks that require flexible and on-demand resource delivery.

This work explores the use of the Hyperledger Fabric platform to power a blockchain-based collaboration model for edge servers in a MEC environment. Task sharing is one of the fundamental problems in regards to MEC networks. However, the literature connected to this topic mostly approaches the problem from the mobile user point of view. In fact, traditional MEC frameworks do not consider task sharing schemes in which servers can independently

coordinate their resources to increase the utilization of the installed computer capacity at the edge level. To the best of our knowledge, there are no current efforts to leverage the idea of blockchain to complement task sharing schemes for servers in MEC networks. The model discussed in this article is powered by the Fabric platform to minimize potential security and privacy breaches. The security traits of our network rely on the permissioned nature of Fabric. The platform implements a membership module in charge of identity authentication and role management in the network. In fact, the module identities to roles and also defines permissions over resources, access to data, and allowed behavior of participants in the blockchain. This functionality prevents external malicious actors to interfere with the normal operation of the task sharing scheme. In addition to that, the enhanced privacy features offered by the network rely on the possibility to implement multiple blockchain channels within a single Fabric platform. Fabric permits the coexistence of several blockchain channels with separate rules, ledgers, and peer-to-peer communication networks within a single Fabric ecosystem. By leveraging this feature, the tasks circulating in the network are kept confidential to edge servers participating in task sharing services. The network restrains non-participating actors from accessing private tasks and their data.

The rest of the paper is organized as follows. Section II presents the background and motivation of this work. We include a brief analysis of blockchain networks, and a revision of the Hyperledger Fabric platform. Section III presents the literature review. Section IV presents the methodology and design elements of the proposed solution. In detail, we include the Hyperledger Fabric framework for task sharing collaboration in MEC networks, the design of the task sharing model at the blockchain level, and the design of the task sharing model at the server level. Section V presents the implementation details of "EdgeChain", the blockchain-based task sharing service demo. Finally, Section VI concludes this paper.

II. BACKGROUND AND MOTIVATION

A. THE PROMISE OF 5G AND BEYOND TECHNOLOGY

The new 5G standard is an upgrade from the previous generation of mobile systems that has promised to improve access, bandwidth, latency, and performance with respect to its predecessor. The 5G standard envisions a communication architecture that offers a very-high bandwidth, very low latency, and the ability to handle a massive amount of subscribers [4]. 5G networks may become the facilitator of new business models with the potential to revolutionize the way we manage, power, and move the economic life of the planet. As a result, new 5G applications will emerge from the infrastructure to serve business models such as autonomous vehicular networks, smart manufacturing, smart grids, e-health services, tactile Internet, immersive entertainment among many others. The industry and academia are currently making huge efforts to finalize the standard so that commercial 5G networks can be deployed massively around the world in less possible time. A Cisco report shows that by the end of 2022 the number of devices connected to the Internet will reach 28.5 billion and the global data traffic will reach 122 exabytes (10^{18}) per month, 64% of that amount corresponding to mobile and wireless systems [5]. An *International Data Corporation* (ICD) report also shows that the projected cumulative financial investment in 5G technologies will also peak in 2022 reaching 370 billion dollars [6]. These numbers can only mean one thing: market conditions are favourable and 5G networks are at the doorstep, ready to make a big entrance in our lives.

The 5G New Radio (5G NR) is the global standardization of 5G networks that is currently being developed by the 3rd Generation Partnership Project (3GPP). The standardization derives three relevant use cases for 5G networks: (1) Enhanced Mobile Broadband (EMBB), (2) Ultra-Reliable Low-Latency Communication (URLLC), and (3) Massive Machine-Type Communications (MMTC). Also, the standard states that 5G will not only support communication, but also content delivery, computation, and control functions [7]. The paradigm shift will power a surging marketplace for products and services characterized by massive volumes of data and whooping requirements of processing power. To meet the demands, 5G relies on several technologies from different fields, such as MEC, millimeter Wave (mmWave), Non-orthogonal Multiple Access (NOMA), Dense Heterogeneous Networks (HetNets), Machine Learning (ML), and Energy Harvesting (EH) to name a few. MEC in particular is a major enabler of 5G networks because it leverages the idea of distributed computing to locate cloud services (i.e., processing power, memory, storage, and control functions) near end users. The architecture introduced by MEC might increase the available data rate and reduce the latency experienced by subscribers using the infrastructure of 5G networks. A detailed explanation of MEC technology is presented in the next section.

B. BASICS OF MEC

MEC is the evolution of the *Mobile Cloud Computing* (MCC) paradigm that relocates cloud services such as processing power, memory, storage, and control functions, spatially close to end subscribers. In mobile systems, this can be done by deploying intelligence capabilities to the edge of the network within the RAN premises [8]. The intuitive idea behind MEC is that the relocation of cloud services might improve the QoS and overall performance of the network. The distribution of intelligence across the network is vital to deploy highly available, highly reliable, and high-performance applications that can respond to users in near real-time, a frequent requirement for upcoming 5G-based business models.

In 2014, Cisco coined the term *Fog Computing* to label their conceptualization of *Edge Computing* architectures. Cisco's Fog describes the migration of cloud computing services from the core to the edge of a network that creates a

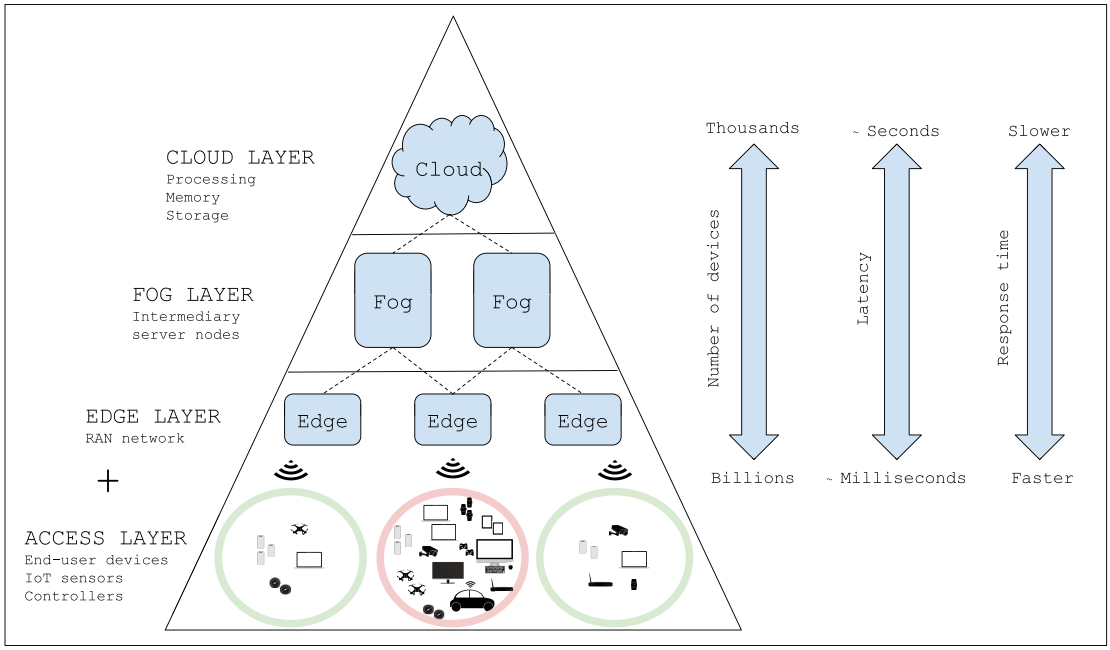

FIGURE 1. Illustration of the Fog/Edge Computing paradigm.

metaphorical fog of distributed computing infrastructure right outside cloud data centers [9]. *Fog computing* deploys processing power, memory, and storage resources in locations geographically close to subscribers so that they can execute computing-intensive applications in fog nodes outside the cloud level. This arrangement reduces latency, congestion, and infrastructure costs in the network. Although fog nodes extend cloud infrastructure and services into the edge, they cannot act as self-managed cloud data centers. Indeed, coordination between the central cloud and the fog is still needed for the proper operation of the system. *Fog Computing* may provide services to mobile users, yet, it is not integrated into the mobile network architecture. In fact, *Fog Computing* nodes are typically owned by private service providers separate from mobile network operators.

To upgrade *Fog Computing*, designers attempted to deploy cloud services even deeper into the edge and gave birth to the idea of MEC. In the upgraded architecture, MEC nodes provide cloud services to the edge of mobile networks, and *Fog Computing* nodes act as intermediaries between edge nodes and the centralized cloud [10]. Fast forward to recent years, the evolution of MEC technology has been driven by many factors including the increasing number of mobile users, huge volumes of data, and the need for high-bandwidth and low latency infrastructure that may serve 5G-based business models. In general, MEC systems may be described by four technical characteristics [11]. First, MEC networks operate on-premises. They can run isolated from the rest of the network and have access to local resources. Second, MEC networks provide proximity. They are deployed geographically close to end users and may collect (and process) as much data as possible from them. Third, MEC networks supply low latency infrastructure with the potential to shorten communication and propagation times. And fourth, due to proximity, MEC networks are capable of providing subscribers with precise location services and contextual information about network conditions for optimization purposes. An illustration of the *Fog/Edge Computing* paradigm is presented in Figure 1. In preparation for the integration of MEC into the 5G standard, 3GPP included MEC in the technical specification report TS 23.501 published by the group in 2019 [12]. The report defines 5G network functions, their roles, and how they can seamlessly interact with the MEC reference architecture. For further explanation on this topic, we invite the reader to explore the details of the report. The MEC architecture stands on four key principles that are summarized in the following subsections.

1) RESOURCE MANAGEMENT

The administration of network resources is a critical component in MEC architectures. The tension between constrained infrastructure and the increasing demand for resources makes allocation methods an important research topic in the communications community. The allocation of resources may have multiple objectives and it is subject to the heterogeneity of the network. More specifically, it should account for a massive number of heterogeneous devices with individual demands, connecting mediums with different characteristics, and running applications of different nature [7]. In that context, task sharing is one of the fundamental problems in MEC networks although most of the time it is only approached from the mobile users point of view. However, a new trend for task sharing focused on devices closer to the core of the

network is emerging rapidly, and that is precisely the area explored in our work.

2) MOBILITY

Emerging network architectures such as the Internet of Things (IoT) come with the need to accommodate a massive number of subscribers that can move across multiple service areas. User mobility in MEC networks may result in frequent handovers that might cause service disruption and affect the overall performance of the system. For example, a mobile user can change coverage areas in the middle of a task sharing service. In that scenario, MEC networks must be capable to provide fast and reliable handover mechanisms that account for changes in the location of the user, dynamic conditions of the access network, and time-varying computation resources [13].

3) HETEROGENEITY

Edge servers may be deployed in different places within the RAN premises to provide coverage to end users with heterogeneous characteristics and requirements [14]. Given that scenario, MEC networks should provide seamless integration of edge devices with the existing access and core infrastructure. Edge servers are deployed in the network to boost available bandwidth and reduce the latency experienced by user applications. However, the processing capacity of the cloud is still way more robust than that installed at the edge layer. Therefore, it may be a good strategy to let the cloud handle tasks that are tolerant of delays to further optimize the use of computing resources at the edge layer. In that direction, MEC systems must provide means for efficient interaction of the cloud and edge layers so that the two instances can coexist and cooperate to the benefit of the network.

4) SECURITY AND PRIVACY

Despite the many advantages of MEC networks, some security and privacy downsides must be addressed. First, due to the proximity of edge servers to end users, traditional MCC security methods are not compatible with MEC architectures. And second, mobile-edge or edge-edge task sharing schemes may be insecure especially when they are executed over wireless transmission channels. Although the integration of cryptography algorithms might help solve security issues, the added complexity can be a problem in terms of propagation and execution delays. Fortunately, emerging technologies such as blockchain may help mitigate these issues [15]. Precisely, this work explores the use of the Hyperledger Fabric platform which is a lightweight cryptography-based network that offers blockchain services to help address security and privacy vulnerabilities in MEC networks.

C. BLOCKCHAIN TECHNOLOGIES AND THE FUTURE OF DECENTRALIZATION

Blockchain is an emerging technology that is revolutionizing industries across the globe due to its unique ability to power decentralized systems. In short, a blockchain is a peer-to-peer computer network that uses sophisticated cryptography and a consensus mechanism to maintain a distributed database on a set of nodes that do not trust each other [16]. As a result, the nodes in a blockchain network can interact without the need of a central entity dictating the rules of the system. Blockchain technologies were first introduced to the public in 2008 with the appearance of Bitcoin, the first ever known distributed payment system, that uses blockchain as its underlying operating platform [17]. To date, the communications industry is being disrupted by the appearance of new management frameworks for future massive networks that move from centralized to decentralized network administration. Blockchain might be an option to take on this scenario due to its ability to power the implementation of decentralized, self-regulated, secure, and intelligent networks.

Blockchain is a peer-to-peer network that uses advanced cryptography to maintain a distributed database of records (i.e., ledger) among a group of independent nodes that do not trust each other. The nodes in a blockchain can interact with the network in the form of transactions validated by a communal verification process known as consensus that normally involves a computing-intensive algorithm. Consensus might be considered a form of digital democracy that allows the nodes in the network to make collective decisions about transactions without the supervision of a central authority. Transactions are packed together inside a structure of data blocks connected chronologically that resemble the form of a chain. By keeping a local copy of the ledger, the nodes in a blockchain become guardians of the history book of the network. This feature provides a high level of transparency and accountability to the system.

Blockchain networks derive directly from Distributed Ledger technology (DLT) that is a database paradigm that defines a set of cryptographic protocols that support the operation of distributed transaction records. The DLT paradigm defines a way to access, validate, update, and manage decentralized databases [18]. The protocols guarantee the immutability, consistency, availability, and transparency of the database records. An illustration of the differences between Centralized Ledger Technology (CLT) and DLT is shown in Figure 2. As a general rule, blockchain networks must enforce three fundamental characteristics: (1) immutability of the distributed ledger, (2) transparency of information and (3) consistency of data across the nodes [19]. In recent years, blockchain technologies have been tailor-designed to serve applications in a variety of economic sectors such as finance, supply chain, energy, etc. The ability of blockchain to power networks with decentralized governance, immutable data, and transparent information has been a key motivation for the development of those solutions [20]. To date, research projects regarding blockchain integration in engineering systems are very popular. We predict that this trend will continue to grow due to the blockchain's potential to revolutionize the way we design future engineering systems.

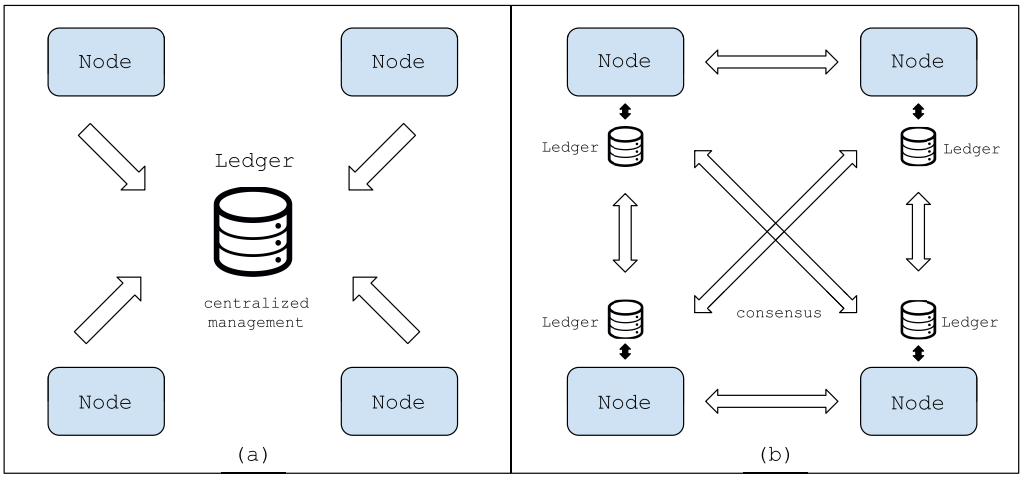

FIGURE 2. (a) CLT paradigm, (b) DLT paradigm.

Blockchain networks can be classified in many forms depending on multiple factors such as type of governance, consensus mechanism, rewarding system, code-base design, membership management etc [21]. The membership category, in particular, can further divide blockchain networks into three different types: Public, Private, and Consortium blockchains. This division is of significant interest to us because it provides core ideas for the construction of the blockchain-based task sharing framework presented in this work.

Public blockchains are un-permissioned platforms with no restrictions to access the infrastructure. In fact, anyone with an Internet connection can join the network and become an authorized node. Participants in public blockchains may have transparent access to data, invoke transactions, and participate in mining activities like transaction verification and block creation. Public blockchains typically have open source design and use consensus mechanisms with high levels of decentralization such as Proof of Work (PoW) and Proof of Stake (PoS) [22]. One of the upsides of public blockchains is that they do not abide by the concept of ownership. In fact, if the founding node leaves the system, the network will continue to run on the remaining nodes with the same operational characteristics. On the other hand, public blockchains are slow and do not scale very well due to the use of hyper democratic consensus to guarantee high levels of decentralization. Popular applications that make use of public blockchains are decentralized payment systems and cryptocurrencies. However, public blockchains may also be a good fit for applications that require fully decentralized, transparent, notarized trust models for open collaboration among the nodes of a network.

Private blockchains are permissioned platforms ruled by a single node inside the network. Likewise their public counterparts, private blockchains operate as peer-to-peer networks, however, the ruling node is in charge of establishing roles for participants that may limit permissions over resources, access to information, and allowed behavior within the network [23]. Private blockchains are invite-only systems meaning that they typically use a membership service that enforces some kind of access control in the network. On the upside, private blockchains may offer higher transaction throughput compared to public blockchains due to their hierarchical role-based structure. On the downside, private blockchains may also have very limited use cases because of the use of highly centralized consensus that is counterintuitive to the idea of blockchain networks. The source code used in private blockchains is typically closed and proprietary meaning that the participants are unable to audit it. Private blockchains are a good fit for applications that require cryptographic services in combination with access control schemes where transparency is not necessary and the ruling central node can be fully trusted. As a side note, private blockchains operate on a much smaller scale than public blockchains, typically inside a single organization.

Finally, consortium blockchains are platforms with a combination of features that resemble both, public and private blockchains. In consortium blockchains, a set of nodes belonging to multiple organizations (possible competitors with different operational logic and incentive structures) join together to form a decentralized network with a role-based structure and distributed control over access, permissions, security, and resources within the network [24]. The governance of consortium blockchains is shared among the participating organizations moving away from the centralized management approach used in private blockchains. Consensus protocols are controlled by predefined consensus nodes that validate transactions and create new blocks. The consensus nodes implement mechanisms to enforce data consistency that are normally more efficient than those available in public blockchains. The design of consortium blockchains is typically open source with a modular architecture that offers plug-and-play components and services that adapt easily to many distributed solutions. Consortium blockchains are popular in many industrial applications such as supply chain, healthcare, banking, and more recently communications

TABLE 1. Public vs. private vs. consortium blockchains.

	Public	Private	Consortium
Decentralization:			
Social analogy:	Libertarian society	Dictatorship	Federated democracy
Access control:			
Access Permission:	Un-permissioned	Permissioned	Permissioned
Membership service:	No	Yes	Yes
Ranked members:	No	Yes	Yes
Consensus:			
Governance:	Distributed	Centralized	Federated
Type of consensus:	Proof-based	Voting-based	Voting-based
Computation demand:	High	Low	Low
Energy demand:	High	Low	Low
Tx throughput:	Low	High	High
Tx latency:	High	Low	Low
Scalability:	Low	High	High
Transparency:			
Data:	Public	Private	Private
Tx visibility:	Visible	Partially visible	Partially visible
Participant's identity:	Anonymous	Disclosed	Disclosed

engineering. A comparison between Public, Private, and Consortium blockchains can be found in Table 1.

D. THE HYPERLEDGER FABRIC FRAMEWORK

Hyperledger Fabric is a consortium blockchain that provides membership, consensus, ledger, and smart contract services for the implementation of multi-purpose decentralized solutions [25]. The technical design of Fabric relies on (1) an open source approach, (2) modular architecture, (3) identity-based roles, and (4) flexible programmable logic that make it stand out from other blockchain platforms. First, Fabric is open source, meaning that the copyright owners grant users the right to use, modify, and distribute Fabric code without restrictions. There is a community of more than 35 partner companies and thousands of developers around the world working on exploiting Fabric benefits. Second, Fabric has a modular architecture with configurable membership, consensus, ledger, and smart contract services that may fit a wide variety of business cases. Third, Fabric uses identity-based roles meaning that the nodes in the blockchain hold digital identities that are mapped to permissions over resources, access to data, and allowed behavior in the network. This feature brings a level of trust among network participants even though some of them might be market competitors with antagonistic interests and inherently untrustworthy relationships. And fourth, Fabric offers flexible smart contracts that can be programmed using general-purpose languages such as *Java, Go, and Node.js*. This is different from other platforms that may use their specific languages for smart contract code (i.e., Ethereum's Solidity programming language). In the following subsections, we discuss the core elements of the Hyperledger Fabric model: (1) membership, (2) consensus, (3) ledger, and (4) smart contracts.

1) MEMBERSHIP

Hyperledger Fabric is a permissioned blockchain in which participant nodes must hold a known digital identity. Identities are assigned to every network component, namely, organizations, peers, and client applications. Fabric makes use of a Public Key Infrastructure (PKI) framework to create, use, store, and revoke cryptographic identities. A PKI is a combination of hardware, software, and advanced cryptography that assigns a pair of public/private keys to network participants through an enrolment process that results in the creation of digital certificates. The x.509 standard is the most common format for digital certificates in PKI systems. PKIs provide trust services for secure communication over the Internet and are widely used in applications such as online e-commerce, banking, and confidential messaging. The Hyperledger Fabric framework uses a PKI-based membership service to verify the identities of participants in the blockchain network. The participants (i.e., administrators, peers, and client applications) are required to have a public-private key pair issued by a trusted CA to sign transactions submitted to the network. As a result, Fabric can grant identity-based permissions over resources, access to data, and allowed behavior in the network. The permissioned nature of Fabric is an attractive option for application scenarios where security and privacy are core to the use case being implemented.

2) CONSENSUS

The core idea of DLT is that network records must be kept distributed among the participant nodes. To achieve that, it is necessary to use mechanisms that can verify and order the records that are added to the ledger. A consensus mechanism is a transaction validation algorithm to reach agreements on the data circulating in unreliable networks. Consensus must keep track of data flows, build transaction blocks, and give assurance that the data in the network is shielded from security threats. Consensus in the Hyperledger Fabric is blockchain operates on a deterministic structure based on endorsements and signature verification for ordering and committing transaction events. At the peer level, an endorsement is a validation made by endorsing peers that consists in the simulation of transaction proposals using their copy of the chaincode. If the simulation produces the same output as the proposal, endorsing peers then sign the proposal and return the signed transaction to the client that originated it. At the ordering level, endorsed transactions are received and validated against the endorsement policy in place. If the transaction complies with the policies, it is declared valid and the consensus process continues with transaction ordering, block creation, and broadcast of new blocks to committing peers.

3) LEDGER

The ledger in a blockchain is a physical database that holds the history of the network and stores facts and data of objects circulating inside it. Blockchain objects and their data may change through the execution of transactions, however, transaction records are unique and immutable. A Fabric ledger has two components: (1) the world state, and (2) the blockchain database. The world state holds facts (i.e., states) of objects in the Fabric network. The states are key-value (KV) pairs that are updated by the ordering service every time a transaction has completed its lifecycle. Client applications can submit transaction proposals to query, update, and delete states in the world state, however, only accepted transactions will induce changes in the database. On the other hand, the blockchain database keeps the historical records of all transactions (valid and invalid) in the Fabric network. In other words, the blockchain database holds the record of how Fabric objects arrive at their current states.

4) SMART CONTRACTS

Smart contracts are immutable pieces of software running on a blockchain that define objects in the network and instructions on how to modify them. They are typically installed, stored, and approved by blockchain participants before they are ready to be used. Smart contracts follow through the business logic of a blockchain application, specifically, they determine how data and transactions are represented in the network. In Fabric context, smart contracts implement the terms, definitions, and processes that rule the interaction of network participants. Smart contracts may submit transaction proposals to query, create, update, or delete states of an object in the network.

E. MOTIVATION

With the popularization of Open-RAN and Network Function Virtualization (NFV) technologies in 4G and 5G systems [26], the current trend in regards to MEC deployment tends to favor the utilization of vendor-neutral Commercial Off-The-Shelf (COTS) servers at the edge level [27]. Typically, each COTS unit has a limited amount of resources in terms of computing power, memory, and storage capabilities. In real implementations, MEC networks are often filled with a high number of low-performance COTS nodes that may struggle to deal with a massive number of requests during peak load hours. In fact, the utilization of network resources at the edge level may still be unbalanced to a great deal when compared to their total installed capacity. There is an increasing interest to develop efficient mechanisms for resource administration at the edge level so that the vision of MEC networks can be fully exploited. We believe that MEC networks can benefit from the notarization, ownership, and chain-of-custody services offered by the Hyperledger Fabric blockchain to address security and privacy concerns related to collaboration mechanisms for edge computing servers.

In this paper we propose a vision that integrates the open standards and management technology of the Hyperledger Fabric blockchain with the architecture of MEC to put forward a powerful, lightweight, and extensible technology foundation for secure management of computing resources in MEC networks. Our framework is built on top of the Fabric platform for a few reasons. First, Fabric is a permissioned blockchain that guarantees identity verification of network participants before granting access permissions to the network. Second, Fabric allows peer-to-peer communication between network nodes that eliminates the need for a central intermediary with control of network interactions. Third, every node holds a local ledger that contains transaction records. The ledger content may be different across the network depending on the access rights granted to a node. More specifically, only nodes with the same access rights can share a ledger with a particular set of transaction records. And fourth, Fabric uses smart contracts to submit transaction proposals to the network and a notary service to avoid double-spending problems.

The objective is to provide a Service Oriented Architecture (SOA) for secure and private task sharing collaboration specifically designed for MEC servers. To achieve that, edge server nodes are instrumented in such a way that Hyperledger Fabric is responsible for their monitoring and management. The system is envisioned to discover servers dynamically, allow peer-to-peer communication, and provide the logic for task sharing services through the execution of smart contracts. The proposed MEC framework will support decentralized computing services with dynamic task scheduling and unified management of resources while maintaining

security capabilities with a special focus on the integrity and confidentiality of tasks and their data. Our task sharing framework will rely on two key principles:

1) *Node Autonomy:* edge servers can perform autonomous connection, discovery, learning, and task sharing through the integration of Hyperledger Fabric services in the network.
2) *Open Collaboration:* edge servers can share tasks with other servers or the cloud to provide elastic networking, computing, and storage capabilities within the network.

The relationship between any pair of edge servers will change from master-slave to equal partnership through the peer-to-peer communication nature of Fabric. Also, the nodes selected to execute tasks in the network will be dynamically chosen based on the amount of resources available for that purpose. The selection procedure will involves a series of optimization routines running at the busy server end. The cloud level will have an overview of the candidate nodes that volunteer to accept tasks. The model leverages Fabric privacy features so that tasks themselves will not be exposed to all candidate nodes or the cloud. Rather, they will only be shared between busy servers and the nodes selected to complete the tasks.

III. LITERATURE REVIEW

As we move towards next-gen communication networks in which massive machine communication, ultra-low latency, and enhanced broadband capabilities will be the dominant trend, the complexity of the emerging standard will prompt fundamental engineering problems related to resource management, heterogeneous collaboration, access control, and network security models. The integration of blockchain technologies with next-gen networks may bring to the picture prime features of blockchain such as decentralization, self-governance, tamper-proof data, transparency, and security attributes to solve some of the problems mentioned earlier. In this section, we review current research work related to the intersection of blockchain technologies with communication networks and the impact that distributed ledgers may have on the design of network architecture, network services, and user applications for the envisioned 5G standard.

A. BLOCKCHAIN AND NETWORK ARCHITECTURE

Blockchain-based architecture for 5G networks opens new possibilities for the implementation of new architecture models that may fulfill the specifications of the standard. Blockchain networks may bring intelligence and cryptographic services into architecture technologies such as Network Virtualization, Software-Defined Networking, Cloud Computing, and Edge Computing models. In the cloud computing department, the authors Zhou et al. proposed a *Clean-room Security Service Protocol* based on a consortium blockchain that monitors the implementation of illegal software on the user's end in a cloud computing setting. The authors claim that their protocol reduces the security threats of the network and has major potential in trusted network computing systems [28]. Next on, the authors Sharma et al. proposed a blockchain-over-cloud solution designed for cloud service operators so they can share their computing resources through smart contracts implementation [29]. Also, the authors Ricci et al. proposed a decentralized cloud storage mechanism based on STORJ blockchain platform. The mechanism breaks down the files that need storage services and then distributes the pieces of data to multiple nodes in the blockchain [30]. Finally, the authors' Xu et al. designed a machine-learned-based energy management framework embedded in blockchain smart contracts to minimize the energy consumption of cloud data centers. The authors test their system on Google cloud clusters and show that their proposed system is able to reduce the energy cost of the data center when compared to other benchmark approaches [31].

Blockchain technologies also present an opportunity to introduce improvements into edge computing architectures. For example, the authors Kotobi and Sartipi propose a blockchain-based architecture for wireless networks in smart city environments [32]. The system introduces a caching mechanism to enhance the communication capabilities of the network and enable massive data collection for smart city applications. A distributed ledger is used to secure communication between smart city applications and IoT sensors deployed in the network. Moving on, the authors Rawat et al. designed a blockchain-based architecture for IoT networks that runs on a wireless network virtualization model that allows multiple virtual network operators with different resource requirements to be accommodated in a blockchain-based ecosystem. The model leverages edge computing to provide processing and storage services to limited IoT devices in the network. The authors focus their analysis on the double spending problem of wireless resources (i.e., frequency) for which they propose a blockchain-based solution approach [33]. Finally, the authors propose a decentralized data management protocol that implements a distributed access control manager for end users in mobile edge networks [34]. The authors make use of blockchain and off-blockchain data storage to give end users decentralized control of their data without the need for a third trusted party in charge of storage services. A summary of the papers discussed in this subsection is presented in Table 2.

B. BLOCKCHAIN AND NETWORK SERVICES

Efficient administration of available infrastructure in communication networks must balance tensions between the growing demand for resources (forced by the massive number of connected devices) and the limited infrastructure owned by network operators. As seen in earlier sections, blockchain can provide a trust model for resource collaboration among a consortium of network operators. The authors

TABLE 2. Application of blockchain technologies in next-gen network architecture.

Use case	Focus	Contribution	Authors
Security	Cloud Computing	Blockchain-based *Clean-room Security Service Protocol* for software implementation in cloud settings.	L. Zhou, G. Wang, and X. Xing
Resource Management	Cloud Computing	Blockchain-based solution for cloud datacenters to share computing resources over a smart contract-enabled mechanism.	S. Sharma, L. Ahuja, and D. Goyal
Storage	Cloud Computing	Decentralized cloud storage mechanism based on the STORJ blockchain platform.	J. Ricci, I. Baggili, and F. Breitinger
Energy Management	Cloud Computing	Smart contract-based energy management framework to minimize energy consumption in cloud data centers.	C. Xu, K. Wang, and M. Guo
Smart Cities	Edge Computing	Blockchain-based caching mechanism for processing massive data in smart city wireless applications.	K. Kotobi and M. Sartipi
IoT Networks	Edge Computing	IoT architectrure based on edge computing and blockchain that implements a wireless virtualization model for network operators.	D. Rawat, and Md. Parwez, and A. Alshammari
Data Management	Edge Computing	Decentralized data management protocol for mobile edge networks.	G. Zyskind, D. Zekrifa, A. Pentland and O. Nathan

Mafakheri et al. presented a blockchain-based distributed authentication mechanism and infrastructure sharing model for small cells in a 5G network [35]. They use a consortium blockchain to store user subscription information in a distributed ledger maintained by the core nodes of the participant operators. In this way, the operators have a distributed access control list to securely provide services to subscribers in the consortium. The proposed model aims to reduce Capital Expenditure (CAPEX) and Operation Expenditure (OPEX) indexes and optimize the economics of the consortium. In the same realm, the authors Rawat et al. proposed a spectrum sharing model for virtual wireless networks based on blockchain that allows primary spectrum owners to lease their available resources to virtual mobile operators [36]. The blockchain network provides a trust model for device-to-device interactions of primary spectrum owners and virtual mobile operators. Next on, the authors Ling et al. proposed a blockchain-based radio access network (BRAN) with a decentralized architecture to manage access and infrastructure sharing in mobile networks [37]. The BRAN architecture uses smart contract logic for network authentication and a variation of PoW consensus based on the identity of the participant to validate interactions between nodes. The authors present a set of test results that suggest that their decentralized model performs better than the centralized counterpart in terms of latency and throughput provided by the network.

The authors Kotobi et al. proposed a medium access control protocol based on blockchain in which multiple cognitive networks compete in an auction game for a priced portion of the available spectrum. They introduce the use of a cryptocurrency in the network for spectrum transactions and a distributed database that is visible to all participants with the intention to enforce transparency in the auction mechanism [38], [39]. Moving on, the authors Fukumitsu et al. proposed a distributed online data storage mechanism based on a peer-to-peer network that does not need central storage services. In the proposed model, the data generated by users is divided into many parts and distributed to the blockchain network using anonymous communication so that no data remains in the user node. The security features provided by blockchain prevent attacks from malicious nodes who may want to tamper with user information stored in the blockchain ledger [40]. Similar to that, the authors Wang et al. propose a decentralized storage model framework for cloud datacenters based on Ethereum blockchain and attribute-based encryption. They presented a demo implemented on the Ethereum test network Rinkeby to show that their proposed scheme is feasible. Finally, next-gen networks can leverage the cryptographic nature of blockchain for security and privacy services. In this context, the authors Kroonmaa et al. filed a U.S. Patent for a data authentication system based on a distributed hash tree infrastructure supported by blockchain. The model uses a blockchain-based hash tree structure formed

TABLE 3. Application of blockchain technologies in network services.

Use case	Focus	Contribution	Authors
Network Economics	Resource Management	Infrastructure sharing model based on consortium blockchain for network operators. The model optimizes CAPEX and OPEX indexes of the network consortium.	B. Mafakheri, T. Subramanya, L.Goratti and R. Riggio
Network Economics	Spectrum Management	Spectrum leasing model based on blockchain for virtual network operators.	D. Rawat, Md. Parwez, and A. Alshammari
Infrastructure Sharing	Resource Management	Blockchain-based ratio access network (BRAN) for secure access control and resource sharing in mobile networks.	X. Ling, J. Wang, T. Bouchoucha, C. Levy and Z. Ding
Network Transparency	Spectrum Management	Auction-based mechanism for spectrum sharing in mobile networks. The model runs on a blockchain network with a cryptocurrentcy model for spectrum transactions.	K. Kotobi and S. G. Bilen
Data Storage	Storage Management	Distributed online data storage mechanism based on a peer-to-peer network. The model breaks down user data in several parts that are stored in a distributed ledger maintained by the network.	M. Fukumitsu, S. Hasegawa, J. Iwazaki, M. Sakai and D. Takahashi
Data Storage	Storage Management	Storage model for cloud datacenters based on Ethereum blockchain for secure data encryption.	S. Wang, Y. Zhang, and Y. Zhang
Network Security	Authentication Management	US Patent: Data authentication system based on a distributed hash tree infrastructure supported by blockchain.	A. Kroonmaa, A. Buldas, and J. Pearce

from digital input records that reduces the possibility to make unauthorized changes in the data structure. A summary of the work discussed in this section is presented in Table 3.

C. BLOCKCHAIN AND NETWORK APPLICATIONS

5G networks open the opportunity for developing new end-user applications with the potential to become game changers for many economic sectors. On the other hand, blockchain technology could be the facilitator to solving some of the challenges related to the implementation of 5G applications, especially in areas such as decentralized management, security, and privacy. For example, the authors Biswas and Muthukkumarasamy proposed a blockchain-based security framework for IoT applications in smart city ecosystems [41]. Their framework runs on a four-layer model that includes a database layer for the integration of blockchain ledgers. The blockchain network serves as a common platform for secure and distributed interaction of smart city devices. The authors claim that their framework could deliver the grounds for direct collaboration between citizens and local governments. Next on, the authors Li et al. introduce a decentralized on-demand energy supply model based on microgrids that provides energy to miner devices in blockchain-based IoT networks [42]. The energy allocation problem is modeled as a Stackelberg game to find optimal profit strategies for both the microgrids and the miners. The proposed system is supposed to alleviate energy limitations for decentralized IoT networks in the presence of different consensus mechanisms.

In the smart industry sector, the authors Fernandez-Caramés et al. propose a smart inventory management system that runs on Unmanned Aerial Vehicle (UAV) infrastructure [43]. The UAVs can scan products with Radio-Frequency IDentification (RFID) tags and send items data to a consortium blockchain that aggregates multiple actors in the supply chain structure. The network validates entries using smart contracts and enforces traceability of the items. The system automates tedious inventory tasks that are typically performed manually by humans. Moving on to smart transportation, the authors Zhang et al. propose a Vehicular Ad-hoc Network (VANET) based on blockchain for self-organizing data transmission for autonomous driving vehicles [44]. The model uses smart contracts installed on a consortium blockchain network to securely process and store vehicle data that feed assisted driving and safety applications. The authors present performance evaluation data that suggest a significant reduction in the consensus convergence time of their blockchain network compared to other traditional

TABLE 4. Application of blockchain technologies in network applications.

Use case	Focus	Contribution	Authors
Secure Smart Cities	IoT Networks	Blockchain-based framework for secure IoT applications in smart city ecosystems.	K. Biswas and V. Muthukkumarasamy
Smart Energy	IoT Networks	Decentralized on-demand energy supply model based on microgrids and blockchain for IoT networks.	J. Li, Z. Zhou, J. Wu, J. Li, S. Mumtaz, X. Lin, H. Gacanin, S. Alotaibi
Smart Industry	UAV	Blockchain-based inventory management model that runs on UAV infrastructure for smart factories.	T. Fernández-Caramés, O. Blanco-Novoa, M. Suárez-Albela, P. Fraga-Lamas
Smart Transportation	VANEC	Blockchain-based VANET for secure and self-organizing data sharing between among autonomous driving vehicles.	X. Zhang and X. Chen
Smart Healthcare	Data Management	Ethereum-based distributed data management model for the healthcare industry.	V. Ramani, T. Kumar, A. Bracken, M. Liyanage and M. Ylianttila

blockchain platforms. Finally, the authors Ramani et al. put forward a distributed data management model for the healthcare industry. The system runs on Ethereum blockchain that provides privacy services for patients' data so that it can only be accessed by authorized parties [45]. The authors present a demo network with the logic of the proposed data model.

D. TASK SHARING AND MEC NETWORKS

In connection to MEC architectures, most of the recent work on task sharing has been approached from the perspective of mobile users. For instance, in 2015, the authors Zhang et al. present a task offloading algorithm for energy constrained mobile users in a cloudlet system that may be intermittently available [49]. The model considers the mobility patterns and load of the user and the availability of cloudlets to make offloading decisions. The authors formulate a Markov Decision Process (MDP) to obtain the optimal task offloading policy for mobile users with the objective to minimize computation costs. In 2016, the authors Xiao et al. proposed an attack-resilient task offloading model for mobile devices [50]. In their work, the authors present a secure mobile offloading game and prove the conditions for the existence of the Nash equilibrium. They also propose a Q-learning based strategy for the mobile users to find stable offloading solutions. In 2018, the authors Luong et al. proposed a deep neural network architecture to make optimal task sharing decisions in a mobile blockchain network [51]. They model the sharing scheme as an auction game between edge servers and mobile users. The computing power of edge servers is the commodity to be auctioned and mobile users must submit bids to *purchase* processing services. The bids become the inputs of the neural network and the outputs determine the task sharing decision along with the associated payments for the computing service.

Also in 2018, the authors Liu et al. studied a MEC-enabled task sharing framework for a blockchain-based video streaming architecture [52]. They design an incentive-based collaboration model for communication systems composed of a macro base station, a set of small-cell base stations, and a set of mobile users all running on a blockchain. In the model, the mobile users have the option to offload computation tasks to nearby small-cell base stations or offload them to a group of device-to-device users to avoid the saturation of macro base stations. The authors formulate the sharing scheme using an adaptive block-size blockchain to minimize the overall latency of the video streaming service. In 2019, the authors Yang et al. propose a dynamic offloading framework for MEC networks [53]. The uplink NOMA is used by multiple users to upload tasks on the same frequency band. The authors formulate the offloading decisions based on a neural network model that jointly optimize the allocation of wireless resources and computational cost.

In their 2020 work, the authors Xiao et al. investigated a blockchain-based task sharing mechanism for mobile users in a MEC network [54]. In their analysis, the mobile users can make task sharing decisions based on the computational performance and latency response of their serving edge server node through a Deep Reinforcement Learning module. The mechanism uses a generic blockchain to keep an immutable ledger of service records which are later used to calculate the money payment earned by the edge servers providing task sharing services. The authors suggest that their approach suppresses the motivation of selfish behavior and faked service record attacks by malicious edge servers. Also

in 2020, the authors Mukherjee et al. study a task offloading model for energy-constrained mobile devices in UAV-enabled MEC networks [55]. In their framework, end users can process tasks themselves or offload them to UAVs that act as access points. The optimal offloading decisions are based on a Deep Neural Network that is trained using Quadratically Constrained Linear Program (QCLP) with Semidefinite Relaxation (SDR) optimal decision generator. Finally, in 2022, the authors Niu et al. present a task offloading scheme for MEC network with Wireless Power Transfer (WPT) and NOMA capabilities [56]. The model optimizes the allocation of computer resources by taking into account the dynamic nature of the wireless channel. The optimal offloading decisions are reached through a Deep Reinforcement learning-based Online Sample-improving (DROS) model that implements a deep neural network that takes channel gains as inputs, and generates the optimal WPT duration as output. Based on the WPT duration, the authors design an optimization algorithm to compute the optimal energy allocation for the offloading data.

In recent years, the focus of task sharing schemes has expanded from models that only serve mobile users to ones that include edge server nodes as well. In 2017, the authors Chen and Xu proposed a collaboration mechanism for small-cell base stations in an ultra-dense user setting [57]. They designed a coalitional game to distribute computation workload among small-cell base stations to summon computing resources at the edge level. The design covers cooperation incentives, fair division of tasks, and a social trust mechanism that helps manage security risks. In 2018, the same authors presented an expansion of their work that aimed to minimize the long-term system-wide latency using Lyapunov optimization. The model proposes an optimization approach that takes into account the capacity and energy constraints of the small-cell base stations [58]. They analyzed two scenarios, one in which a central entity makes task sharing decisions for the small-base stations, and another in which the base stations coordinate their sharing decisions in a distributed style using a coalitional game approach.

In 2020, the authors He and Wang presented a task sharing algorithm for MEC networks that provided a worst-case latency response for latency-sensitive applications in a mobile network [59]. They formulated a task sharing problem for base stations formulated on a stochastic arrival model that maximizes a utility function that depended on the time-average throughput of the system. They built the model based on the assumption that the tasks utilizing sharing services carry on the same workload. This assumption made the problem easier to solve, however, it does not hold in a real network scenario. The same authors upgraded their work in 2021 and presented a task sharing model in which they allow the workload of the tasks to have different profiles [60]. They presented a scheduler and an online algorithm based on Lyapunov optimization theory that provides a worst-case latency response model for the system. The proposed algorithm guarantees that all non-dropped tasks are served within a given latency bound. According to the authors, their algorithm tends to overestimate the worst-case latency of the system. In their simulations, it was found that the latency bound provided by their method was considerably larger than the actual value.

Finally, our work in 2020 proposed a framework for secure and private resource collaboration in MEC networks using the Hyperledger Fabric blockchain platform. The framework was presented as an alternative to minimize security and privacy vulnerabilities in MEC networks [61]. Although our framework was presented in the context of MEC, it can be generalized to cloud/fog computing frameworks. In 2021, we followed up our previous work with a blockchain-based collaborative task offloading model for MEC servers based on the Fabric platform. The model aims to maximize the utility function of the system that depends on the characteristics of the computational tasks and key performance metrics of Fabric [62]. Fast forward to 2021, we further expanded our work with a blockchain-based task sharing model that takes into account the functional dependencies of the tasks [63]. The task sharing scheme is modeled as a multi-processor task scheduling problem that allocates a set of precedent-dependent tasks among a set of available edge server nodes by jointly optimizing the utility function of the system and the makespan of the tasks.

To summarize, the available literature on task sharing schemes for MEC networks can be divided into two branches: (1) task sharing models centered around mobile users, and (2) task sharing models centered around edge server nodes. The articles in [49], [50], [51], [52], [53], [54], [55] and [56] leverage machine learning, game theory, and blockchain technologies to support efficient and trusted task sharing collaboration schemes for mobile users in MEC networks. On the other hand, the articles in [57], [58], [59], and [60] direct their attention to task sharing schemes for base stations in mobile networks that act as edge servers. To the best of our knowledge, our work in [61], [62], and [63] is currently the only effort to explore the use of the Hyperledger Fabric platform to power task sharing collaboration for servers at the edge layer. A summary of the research work mentioned in this section can be found in Table 5.

IV. THE HYPERLEDGER FABRIC FRAMEWORK FOR TASK SHARING COLLABORATION IN MEC

The framework is built under the assumption that a collection of $S+1$ nodes, namely, $\mathcal{S} = \{1, \ldots, S\}$ edge server nodes plus the cloud level, get together to form a consortium in the Hyperledger Fabric platform. At any given time, a busy server node $i \in \mathcal{S}$ submits a task sharing service request to the blockchain network to get additional resources to process pending computational tasks. After the request is in place, the cloud level shortlists a set of candidate servers $\{1, \ldots, j, \ldots, k\} \in \mathcal{S}$ willing to process the pending tasks based on the information provided by the monitoring tool running at this location. With the information in hand, the

TABLE 5. Summary of examined task sharing schemes for MEC networks.

Use Case	Focus	Contribution	Authors
Mobile Users	Cloudlet Networks	Task offloading algorithm for energy constrained mobile devices in cloudlet networks. MDP-based formulation for optimal task offloading policies.	Y. Zhang, D. Niyato and P. Wang [49]
Mobile Users	Game Theory + Q Learning	Attack-resilient task offloading model for mobile devices. Task offloading game with Q Learning based strategies for stable offloading decisions.	L. Xiao, C. Xie, T. Chen, Huaiyu Dai and H. V. Poor [50]
Mobile Users	Blockchain + Neural Networks	Task sharing scheme for mobile users in blockchain-powered MEC networks based on auction game theory and deep neural networks.	N. Luong, Z. Xiong, P. Wang and D. Niyato [51]
Mobile Users	Blockchain	Task offloading framework for mobile users in a MEC-enabled video streaming network based on a generic blockchain with adaptive block size.	M. Liu, F. Yu, Y. Teng, V. Leung and M. Song [52]
Mobile Users	NOMA + Neural Networks	Dynamic offloading framework for MEC networks with NOMA uplink to upload mobile tasks. Offloading decisions are based on a neural network model that jointly optimizes the allocation of wireless resources and computational cost	B. Yang, X. Cao, J. Bassey, X. Li, T. Kroecker and L. Qian [53]
Mobile Users	Blockchain + Reinforcement Learning	Task sharing mechanism for mobile users in a MEC network that runs on a generic blockchain with deep reinforcement learning intelligence.	L. Xiao, Y. Ding, D. Jiang, J. Huang, D. Wang, J. Li and H. Vincent [54]
Mobile Users	NOMA + Deep Neural Networks	Task offloading model for energy-constrained mobile devices in UAV-enabled MEC networks. The offloading decisions are based on a Deep Neural Netowrk trained using a QCLP decision generator.	M. Mukherjee, V. Kumar, A. Lat, M. Guo, R. Matam and Y. Lv [55]
Mobile Users	NOMA + Deep Reinforcement Learning	Task offloading scheme for MEC networks with WPT and NOMA capabilities. The optimal offloading decisions are based on a Deep Reinforcement learning model.	J. Niu, S. Zhang, K. Chi, G. Shen, W. Gao [56]
Edge Servers	Game Theory	Task sharing scheme for small-cell base stations for MEC-enabled ultra-dense networks based on a coalitional game and a fair share incentive mechanism.	L. Chen and J. Xu [57] [58]
Edge Servers	Lyapunov Optimization	Task sharing algorithm for base stations in a mobile network that provides worst-case latency response based on a stochastic task arrival model and Lyapunov optimization.	X. He and S. Wang [59] [60]
Edge Servers	Consortium Blockchain	Collaboration framework for secure and private task sharing in MEC networks using the Hyperledger Fabric blockchain platform.	A. Vera-Rivera, A. Refaey, and E. Hossain [61] [62] [63]

busy server solves an optimization problem to select a few nodes from the candidate list that will become the service providers. After node selection is complete, the task sharing scheme (i.e., upload, remote execution, download of results) is overseen by the blockchain network through the execution of smart contracts available on the platform. The illustration of the proposed framework can be seen in Figure 3. The task sharing request and the sharing scheme that comes after will follow the sequence below.

1) The edge server node $i \in \mathcal{S}$ owns a set of computational tasks that require processing. The server is short of resources and requests the task sharing service to the blockchain network.
2) The cloud level runs a monitoring tool with an overview of the network. It can shortlist a set of candidate nodes $\{1, \ldots, j, \ldots, k\} \in \mathcal{S}$ to take on the tasks owned by server i.
3) With the information provided by the cloud, the busy server i allocates the queued tasks among a few selected candidate nodes. The selection procedure is based on the optimization of particular objectives at the busy server end. The selected nodes become service providers.

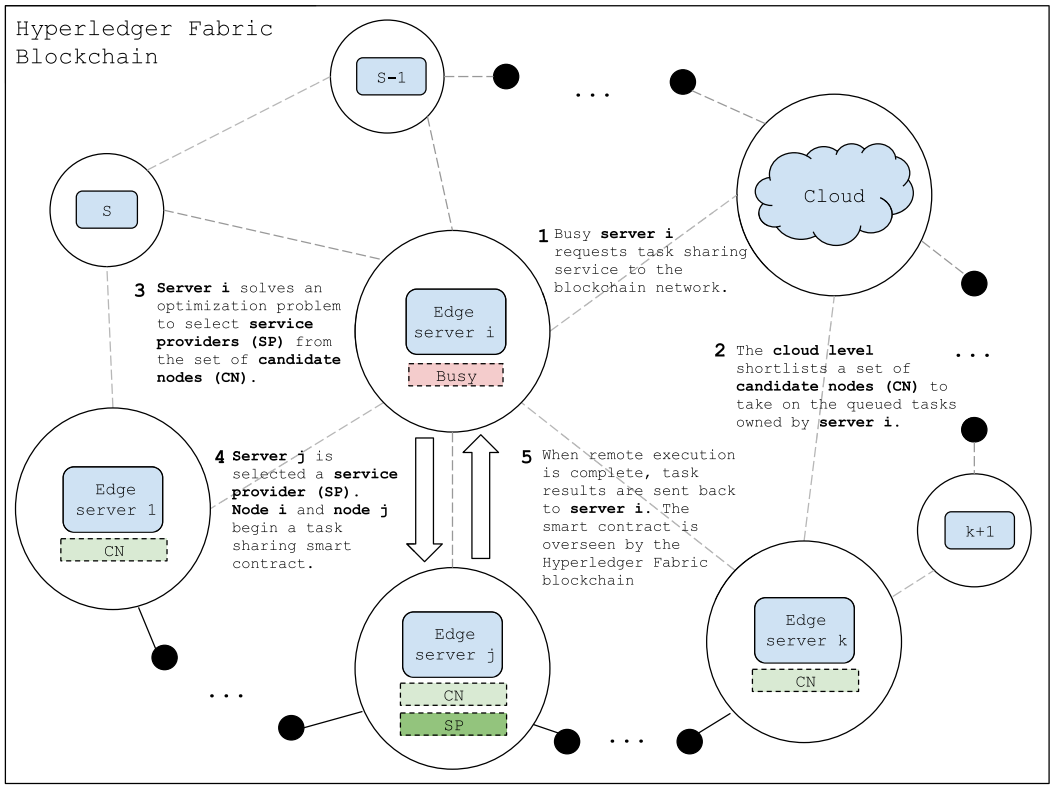

FIGURE 3. Illustration of the Hyperledger Fabric framework for task sharing collaboration in MEC networks.

4) Finally, the busy server *i* and every selected candidate *j* invoke Fabric smart contracts to carry through the sharing process.
5) The sharing process will complete its lifecycle following the rules of the Hyperledger Fabric framework.

It is important to note that the nodes involved in task sharing transactions must follow the ledger rules dictated by the Fabric platform. More specifically, ledgers that receive updates as a result of task sharing transactions are only visible to nodes participating in the associated sharing service. This feature guarantees confidentiality and privacy of tasks and their data during the process. On a separate note, the smart contracts installed on the blockchain will implement the task sharing logic in the system and be responsible for processing transaction requests. Any edge server node can activate a smart contract by submitting an initiation transaction to the network. When the initiation is approved, the smart contract enters into an active state and starts receiving transaction proposals from the server nodes in the network. For an expanded explanation on the Hyperledger Fabric-based task sharing framework, we advise the reader to review our 2020 article *"A Blockchain Framework for Secure Task Sharing in Multi-Access Edge Computing"* [61].

A. TASK SHARING MODEL AT THE BLOCKCHAIN LEVEL

As discussed in previous sections, Hyperledger Fabric is an open-source platform that offers blockchain services for multi-purpose distributed solutions. Fabric offers a modular architecture of peers, membership, ledger, consensus, and smart contracts with enhanced cryptographic attributes for decentralized applications. In our model, a set of edge servers and the cloud level get together to form a consortium in the Hyperledger Fabric platform. A set of computational tasks that require processing services are Fabric objects, and the logic of the proposed sharing scheme is embedded in the chaincode installed on the network peers.

The proposed solution accommodates a Membership Service Provider (MSP) in charge of identity authentication and role management in the network. The players participating in the network (i.e., organizations, peers, and client applications) receive standard *x.509* identities and a public-private key pair from trusted Certificate Authorities (CAs). The MSP maps identities to network roles that define permissions over resources, access to information, and allowed behavior in the blockchain. In addition to the MSP, the network implements an Ordering Service module (OS) in charge of transaction verification and consensus scheme. Finally, the logic of the task sharing scheme is implemented by smart contracts installed on the network peers. The smart contracts are governed by the Hyperledger Fabric architecture and the policies defined by the task sharing logic. Together, they guarantee secure and private communication between nodes involved in the sharing scheme. The dynamics of our task sharing solution can be described as follows:

1) *Service Request:* A resource-constrained edge server node owns a task-flow computational job with a set of precedent-dependent tasks. The server requests the task sharing service to the blockchain network.
2) *Discovery of Candidate Nodes:* Upon receiving the task sharing request, the monitoring tool sitting at the cloud level shortlists a set of candidate nodes that may offer the task sharing service.
3) *Selection of Service Providers:* With the help of optimization tools, the busy server determines the optimal allocation of tasks among the set of candidate nodes. The allocation maximizes the utility function of the server and minimizes the makespan of the schedule of the set of tasks. The allocated servers are called service providers.
4) *Task Sharing Contract:* The busy server and every service provider start a task sharing contract that oversees uploading, remote execution, and return of results of tasks subject to the sharing service. The contracts will use the blockchain services of the Fabric platform.

Right after the Fabric network is launched, the ordering service takes the lead as the first point of administration to set up the network according to the configuration policy agreed upon by participant organizations (i.e., Edge Service Providers (ESPs) and Cloud Service Providers (CSPs)). The configuration policy contains information regarding organizations and peers, the setup of the membership service, and the type of consensus running at the ordering service. Next, the Fabric consortium creates a global channel where the plenary of server nodes and the cloud level may communicate with each other and request task sharing services. The channel implements a distributed ledger of records with the history of submitted transactions to the channel space. Once the global channel is configured, the *Service Terms Chaincode* is installed and committed to the channel. The *Service Terms Chaincode* will track *(i) request, (j) reply, and (k) decision* transactions related to the terms of task sharing services, however, it will not execute the task sharing itself. This feature is an intentional design choice to protect the privacy of the tasks and their data by not publicly exposing them to the network.

The monitoring tool sitting at the cloud level has access to information related to network resources and performance. The module can shortlist a set of available servers that may offer task sharing services to the network. On the other hand, busy servers have an optimization module that can map pending tasks on their end to a few available nodes from the candidate list. Technically, the monitoring and optimization modules (i.e., client applications) are not part of the blockchain, however, they also receive digital identities from CAs and are integrated into the network through the use Fabric APIs. Once integrated, the client applications may use the global channel to submit transaction proposals that follow the logic of the *Service Terms Chaincode*. When a busy server requests the task sharing service, it invokes a request transaction on the global channel. The cloud node then uses its monitoring tool to discover available servers that might become service providers after the selection process. With that information in hand, the busy server uses the optimization module to find the optimal subset of servers from the candidate list that will be in charge of remote task execution. Finally, the busy server and the service providers make use of separate private channels equipped with the *Task Sharing Chaincode* to oversee *(m) task upload, (n) remote execution, and (p) return of results* transactions that complete the lifecycle of the task sharing service. The global and private channels are completely decoupled from one another, meaning that they implement their own policy rules, ledgers, and peer-to-peer communication. The decoupled channels in our model separate the two pieces of chaincode in the network (i.e., *Service Terms* and *Task Sharing* Chaincode) to preserve the privacy of tasks and their data. In fact, under the rules of our model, the tasks are only visible to servers partaking in the sharing service restricting other nodes in the blockchain to have access to this information. The task sharing model at the blockchain level is summarized in Figure 4.

B. TASK SHARING MODEL AT THE SERVER LEVEL

1) INTRODUCTION TO TASK SHARING

Task sharing refers to the partial or full migration of computing-intensive tasks from resource-constrained devices to near infrastructure with plenty of available resources. The migration aims to solve resource shortcomings (i.e., processing power, memory, storage, and energy deficiencies) at strained nodes in a network [46]. Task sharing decisions deal with three basic questions: (1) what to share, (2) where to, and (3) at what time. The optimal answers might be based on specific objectives than depend on network conditions and user preferences, such as installed computing power, available bandwidth, and latency requirements.

Computational tasks can be thought of as units of execution of software applications that perform specific activities within the program. Their properties might be defined by data size, required CPU cycles, release times, completion deadlines, node exclusivity, and functional dependencies. It is possible to analyze the sharing profile of a software application in terms of three aspects: (2) *separability*, (2) *a priori data*, and (3) *dependencies*. First, for applications that enable code or data partitioning, not all the parts are candidates for sharing. For example, input and output events are tasks that typically need to be executed at the mother node, therefore, they most likely cannot be offloaded. Second, for applications that process data, complete *a priori* knowledge of the amount of information to be processed is not always possible, for example, online gaming software that requires continuous execution. For this type of application, task sharing can become a very complex process. Finally, for applications composed of tasks that receive (or provide) feedback from (or to) others, parallel sharing might not be possible. In that

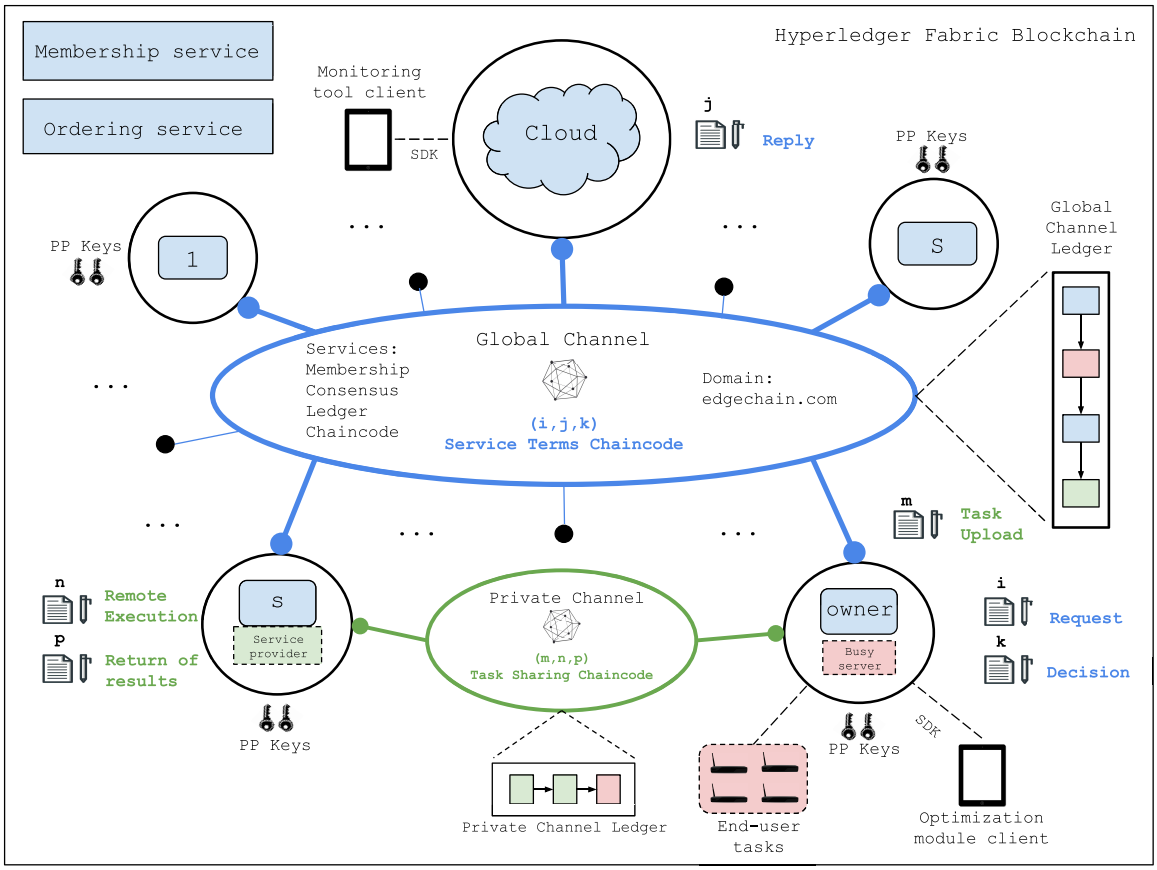

FIGURE 4. Task sharing model at the blockchain level.

case, the dependencies among the tasks may play a decisive role in the sharing decisions.

Depending on their sharing eligibility, software applications may apply three types of task sharing strategies: (1) local execution (no sharing), (2) full sharing, and (3) partial sharing [47]. The details of the sharing modes are explained next.

1) *Local Execution:* the computation of tasks is done entirely at local premises. This may happen when software applications cannot be partitioned into shareable tasks or when task sharing schemes do not bring benefits to the network. The execution delay accounts for time spent on the local execution of tasks.
2) *Partial Sharing:* the computation of tasks is done partially at local premises and the rest is done at other nodes in the network. This happens when software applications are composed of both, shareable and non-shareable tasks. The execution delay accounts for time spent on local execution of non-shareable tasks plus time spent on transmission, remote execution, and return of results of shareable tasks.
3) *Full Sharing:* the computation of tasks is passed entirely to remote nodes in the network. The execution delay accounts for time spent on transmission, remote execution, and return of task results.

In any case, task sharing schemes are strategies to hit certain performance targets defined by the network designers. The targets may involve the maximization of utility functions with user preferences, minimization of the execution time of software applications and their tasks, minimization of energy consumption, or any other system-defined operation. To implement a task sharing scheme, a network must be equipped with three essential modules: (1) task profiler, (2) system profiler, and (3) decision engine [48]. The task profiler is the module that determines what parts of a software application can be shared. It can translate applications into a set of tasks, computing requirements, and dependencies. The system profiler is the module responsible for monitoring the performance parameters of the network. The parameters might include CPU power, storage, memory, consumed energy, etc. Finally, the decision engine is the algorithmic approach to making sharing decisions based on the information provided by the task profiler and the system profiler.

2) MATHEMATICAL REPRESENTATION OF TASK-FLOW COMPUTATIONAL JOBS

As a general rule, computer applications are a combination of software code and relevant data designed to perform specific jobs for end users or other applications. They can be viewed

as a set of computational jobs that need to be sequentially executed to produce the desired output. More specifically, a computational job is an independent unit of execution of a computer application that performs computational work with a specific pre-defined objective and is normally associated with a computer process. Jobs can be further divided into tasks that are smaller execution units that may hold a relational dependency with other tasks inside the same job. Computational tasks can range from a single instruction to complicated code structures that may involve data manipulation and optimization problems. The execution of tasks is normally controlled by a job scheduler and once tasks are finished successfully, the computational job is said to be complete.

One of the fundamental design postulates of software development is known as the Acyclic Dependencies Principle (ADP). ADP states that "the dependency graph of packages or components in a software application should have no cycles" [64]. The principle implies that the relational dependencies among components in software applications must flow only in one direction and may never form closed loops. In that context, a task-flow computational job is a unit of execution of a software application that contains a set of tasks and a collection of data that requires a number of computing resources to be completed. Formally, a task-flow computational job is the tuple $\mathcal{J} = \{\mathcal{A}, \mathcal{G}(\mathcal{A})\}$, where $\mathcal{A} = \{1, \ldots, A\}$ is a set of tasks contained by $\mathcal{J}$ and every $a \in \mathcal{A}$ is portrayed by the tuple $(\delta_a, \rho_a, \tau_a)$ where δ_a is size of the task in bytes, ρ_a represents the CPU cycles needed for task execution, and τ_a is the deadline in seconds. $\mathcal{G}(\mathcal{A})$, on the other hand, is a direct control-flow graph with no cycles that captures the functional relationships (i.e., dependencies) of the set of tasks $\mathcal{A}$ inside the job $\mathcal{J}$. Moreover, the vertices of $\mathcal{G}$ represent the tasks themselves, and the edges represent their precedence dependencies. Every task $a \in \mathcal{A}$ is labeled by its corresponding data size, CPU requirement, and deadline time. At the same time, every edge dependency in $\mathcal{G}$ represents the fact that any task $a \in \mathcal{A}$ must be completed before the execution of the successor task $b \in \mathcal{A}$. Finally, the edge weight $\delta_{a,b}$ represents the size of the data transferred from task a to task b. Figure 5 shows the graph representation of a task-flow computational job with general dependent tasks. The mathematical framework of our model is built upon the assumptions laid out in this section.

3) MATHEMATICAL FRAMEWORK FOR THE TASK SHARING SCHEME

Let us consider a MEC network composed of a set of edge servers $\mathcal{S} = \{1, \ldots, S\}$ and a cloud level all running as independent nodes on the Hyperledger Fabric blockchain. Thanks to a decentralized architecture and the use of advanced cryptography, Fabric provides a trust model for a secure and private task sharing service in the network that does not need a central authority in charge of sharing decisions. Let us also consider a server node from the set $\mathcal{S}$ that owns a task-flow computational job $\mathcal{J} = \{\mathcal{A}, \mathcal{G}(\mathcal{A})\}$ that may be

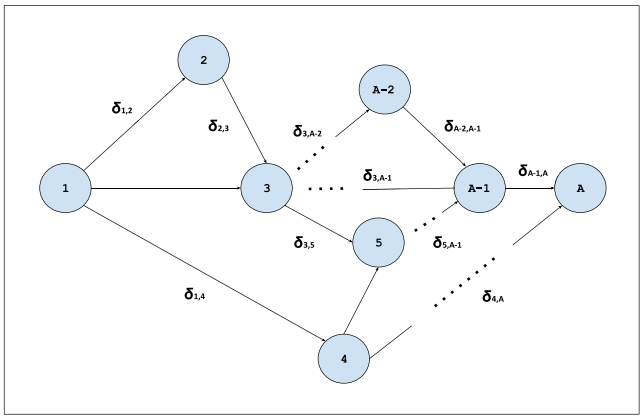

FIGURE 5. Graph representation of a task-flow computational job with general dependent tasks.

decomposed in a set of indivisible tasks $\mathcal{A} = \{1, \ldots, A\}$, and the acyclic graph $\mathcal{G}(\mathcal{A})$. Each $a \in \mathcal{A}$ is the tuple $(\delta_a, \rho_a, \tau_a)$ where δ_a is the size of the task in bytes, ρ_a is the CPU cycles needed for task execution, and τ_a is the deadline in seconds. On the other hand, the acyclic task-call graph $\mathcal{G}$ captures the task precedence dependencies in $\mathcal{A}$. From $\mathcal{G}$, the general task precedence dependency matrix G is defined in equation (1).

$$G^{a,b} = \begin{cases} 1, \text{ if task } a \text{ precedes task } b \\ 0, \text{ otherwise.} \end{cases} \tag{1}$$

Also, from graph $\mathcal{G}$, the immediate task precedence dependency matrix I is defined as follows in equation (2):

$$I^{a,b} = \begin{cases} 1, & \text{if task } a \text{ immediately precedes task } b \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

If a busy server is not able to provide computing services to job $\mathcal{J}$, the server is given the possibility to outsource the execution of the tasks inside $\mathcal{J}$ to available infrastructure in the network. In that case, the busy server can submit a task sharing request to the blockchain asking for computing resources. After the request is in place, the cloud shortlists a set of available nodes $s \in \mathcal{S}$ that are well conditioned to provide task sharing services. The process of shortlisting servers is registered in the availability matrix B defined in equation (3).

$$B^{a,s} = \begin{cases} 1, & \text{if server } s \text{ can execute task } a \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

With the information provided by the cloud, the busy server finds the optimal allocation of tasks among available servers that maximizes its utility function and minimizes the makespan of the tasks. Once a sharing decision is in place, the rest of the task sharing process (i.e., *task upload*, *remote execution*, and *return of results*) is managed by the blockchain through the execution of task sharing contracts. More specifically, if a server with CPU-clock frequency f and hardware constant c is set to process a computational task a locally, the execution time and consumed energy can

be calculated according to equations (4) and (5). The term c in equation (5) is the server's hardware constant introduced by the Dynamic Voltage and Frequency Scaling (DVFS) computation model for MEC servers [7].

$$t^a = \frac{\rho_a}{f}. \tag{4}$$

$$e^a = c\rho_a f^2. \tag{5}$$

On the other hand, if a busy server is set to outsource task a to another server s given that the two nodes share a communication channel with bandwidth bw^s, then the total round-trip time of a is defined in equation (6). The first term o the equation represents the execution time of a in s. The second term is the uploading time of a from the owner to s given that δ_a is the size of the task. The third term constitutes the downloading time of the results of a from s back to the task owner given that $\sum_b \delta_{ab}$ is the aggregated size of the results of a that will be fed into all its immediate successors b. Finally, the fourth term accounts for the communications overhead introduced by the Hyperledger Fabric platform. Specifically, the overhead is the latency injected by the smart contracts in charge of the task sharing process. The equation (8) shows the energy consumed by the owner during the communication process where P is the average power of the busy server.

$$t^{a,s} = \frac{\rho_a}{f_s} + \frac{\delta_a}{bw^s} + \frac{\sum_b \delta_{a,b}}{bw^s} + t_{HFB}. \tag{6}$$

$$e^{a,s} = \frac{\delta_a}{bw^s} P. \tag{7}$$

From equations (4) and (6), we define the execution delay matrix D with elements $D^{a,s} \in \mathcal{R}$ that represent the execution time of the tasks $a \in \mathcal{A}$ by servers $s \in \mathcal{S}$. We also define the communication delay matrix C with elements $C^{a,s} \in \mathcal{R}$ that represent the round-trip communication time of $a \in \mathcal{A}$ from the busy server to any other server $s \in \mathcal{S}$ in the system. The utility that a busy server gets by outsourcing task execution to other server nodes is represented by the weighted average of four ratios. The first two ratios are time and energy savings as a result of the task sharing service; They are defined in equations (8) and (9).

$$r_1 = \frac{t^a - t^{a,s}}{t^a}. \tag{8}$$

$$r_2 = \frac{e^a - e^{a,s}}{e^a}. \tag{9}$$

Moreover, the last two ratios capture the reputation of the server nodes inside the blockchain network. More specifically, r_3 is the transaction success ratio that represents the portion of transactions successfully endorsed by peers compared to the total number of transactions submitted to the network. This ratio can be interpreted as a measure of the reliability of a node in the blockchain network. On the other hand, r_4 is the normalized block size ratio that measures the relative block size used by a peer compared to the maximum block size allowed by the blockchain. As stated in [65], bigger blocks may lead to a reduction of the latency introduced by Fabric. This holds under the assumption that the transaction throughput in the blockchain is not saturated. In short, this ratio is a measure of how fast a server node manages transactions inside the blockchain. At this point, the elements of the utility matrix U are defined as the scalar product presented in equation (10). The entries $U^{a,s} \in R$ are calculated as the weighted average of the performance ratios $\mathbf{r^{a,s}} = [r_i]$. The weight vector $\mathbf{w}$ represents the performance preferences of the busy server. The sum of the elements of $\mathbf{w}$ is equal to one, $\sum_i w_i = 1$.

$$U^{a,s} = \mathbf{w} \cdot \mathbf{r^{a,s}}. \tag{10}$$

The allocation variable $X_{a,s} \in \{0, 1\}$ is defined in equation (11).

$$X_{a,s} = \begin{cases} 1, & \text{if task } a \text{ is allocated to server } s \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The time at which every task must be released for remote execution is represented by R_a. Then, the makespan of the schedule of the tasks, $\mathcal{C}_{max} \in \mathcal{R}$, is defined in equation (12).

$$\mathcal{C}_{max} = \max_{a \in \mathcal{A},\, s \in \mathcal{S}} \left(R_a + \left(D^{a,s} + C^{a,s}\right) * X_{a,s}\right). \tag{12}$$

Also, the aggregated utility of a schedule, $\mathcal{U} \in \mathcal{R}$, is defined in equation (13).

$$\mathcal{U} = \sum_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} U^{a,s} X_{a,s}. \tag{13}$$

Finally, the dual objective function of the task sharing model at the server level, $\mathcal{F} \in \mathcal{R}$, is defined in equation (14).

$$\mathcal{F} = \mathcal{U} - \mathcal{C}_{max}. \tag{14}$$

4) PROBLEM FORMULATION

Scheduling Theory is the branch of mathematics that studies algorithmic solutions for task allocation problems in sets composed of autonomous agents. The task sharing problem analyzed in this section considers a set of autonomous edge servers $\mathcal{S} = \{1, \ldots, S\}$ running as independent nodes on the Hyperledger Fabric blockchain. Some of the servers are available to share their processing resources to execute a computational job $\mathcal{J} = \{\mathcal{A}, \mathcal{G}(\mathcal{A})\}$ owned by one of the agents in the system. The job is composed of a set of discrete computational tasks $\mathcal{A} = \{1, \ldots, A\}$. The tasks may have different sizes, CPU requirements, and deadlines. Also, their precedence dependencies are modeled by the directed acyclic graph $\mathcal{G}$, with no specific release times. The servers may also have different computing power and hardware characteristics. The physical medium and available bandwidth between any pair of servers may be different as well. The joint objective of the problem is to determine an optimal mapping of tasks and servers to maximize the aggregated utility of the busy server and minimize the completion time (i.e., makespan) of the resulting schedule.

As stated by Lawler and Lenstra [66], a collaboration scheme of this type can be modeled as the **R|PREC|F|JP**

multi-processor task scheduling problem. The notation is explained in detail next:

- The unrelated servers condition **R** states that the processing time of any task is a function of the chosen executing server and the task itself.
- The precedence constraint **PREC** assumes that the dependencies of the set of tasks form a directed acyclic graph $\mathcal{G}$. It is also assumed that the tasks may have distinct execution times and also no specific release times. They may be scheduled as soon as their predecessors are completed.
- The function **F** represents the optimal criteria of the model. The dual objective of our task sharing problem aims to maximize the aggregated utility of the busy server and minimize the completion time of the schedule.
- The communication type **JP** states that the cost (i.e., delay) of communicating a task between a pair of servers in the network depends solely on the servers and the task itself.

In short, the problem is set to allocate a finite set of precedence dependent tasks ($\mathcal{A} = \{1, \ldots, A\}$) among a set of available edge servers ($\mathcal{S} = \{1, \ldots, S\}$) with the objective to maximize the dual function ($\mathcal{F} = \mathcal{U} - \mathcal{C}_{max}$). The edge servers and the cloud level run as independent nodes on the Hyperledger Fabric blockchain. The blockchain metrics used in the model capture the health of the nodes in terms of their transaction reliability, and the communication overhead introduced by the blockchain platform. The resources of the system and the performance of Fabric are tracked by a monitoring tool sitting at the cloud level. The monitoring tool provides input information to servers requesting the sharing service so that they can make appropriate allocation decisions. The formulation of the task sharing problem at the server level takes the form of a Mixed Integer Linear Program (MILP) presented in equations (15a). The summary of symbols and notations can be found in Table 6.

The problem formulation has an MILP structure which is by definition a non-convex problem. However, the relaxed version of the model may have convex properties. If this is the case, problem solutions may be approximated in tractable time using well-established methods such as branch-and-bound or heuristic techniques. Constraint (15c) guarantees the one-to-one task-server mapping condition. Constraints (15d) and (15e) force task sharing solutions to available servers offering time and energy savings. Constraint (15f) gives the assurance that a task (a) cannot start before the processing of all its predecessors (b) given that ($I^{a,b} = 1$). Finally, constraints (15g) to (15i) in concert with the overlapping variable (θ) and the constant (M) specify that the execution time of any two tasks (a) and (b) in a server do not overlap given that ($G^{a,b} = 0$).

$$\max_{R_a, X_{as}} \quad \mathcal{U} - \mathcal{C}_{max} \tag{15a}$$

$$\text{s.t.} \quad X_{a,s} \in \{0, 1\} \tag{15b}$$

$$\sum_{s \in \mathcal{S}} X_{a,s} = 1 \tag{15c}$$

TABLE 6. Summary of symbols and notations.

Notation	Definition
Input parameters:	
$\mathcal{J}$	Task-flow computational job
$\mathcal{A}$	Set of computational tasks in $\mathcal{J}$
$a, b \in \mathcal{A}$	Individual tasks in $\mathcal{A}$
d_a	Data size of a
ρ_a	CPU requirements of a
τ_a	Deadline of a
$\mathcal{G}$	Precedence dependency graph of $\mathcal{A}$
$\mathcal{S}$	Set of edge servers in the system
$s \in \mathcal{S}$	Individual server in $\mathcal{S}$
f	CPU-clock frequency of the busy server
c	Hardware constant of the busy server
P	Power consumption of the busy server
t_{HFB}	Latency introduced by HFB*
Indicator variables:	
G	General task precedence dependency matrix
$G^{a,b}$	General precedence of a with respect to b
I	Immediate task precedence dependency matrix
$I^{a,b}$	Immediate precedence of a with respect to b
B	Server availability matrix
$B^{a,s}$	Availability of s to process a
Data variables:	
t^a	Local execution time of a
e^a	Local energy consumption of a
$t^{a,s}$	Outsource round-trip time (a, s)
$e^{a,s}$	Outsource energy consumption (a, s)
$r^{a,s}$	Performance ratios (a, s)
w	Performance preferences
D	Execution delay matrix
$D^{a,s}$	Execution time (a, s)
C	Communication delay matrix
$C^{a,s}$	Communication time (a, s)
U	Utility matrix
$U^{a,s}$	Utility of the sharing service (a, s)
Support variables:	
$\theta_{a,b}$	Overlapping between a and b
M	Default upper bound of the make span
Decision variables:	
$X_{a,s}$	Allocation variable (a, s)
R_a	Scheduled release time of a
Objective functions:	
$\mathcal{U}$	Aggregated utility of the task schedule
$\mathcal{C}_{max}$	Make span of the schedule
F	Objective function of the task sharing model

* HFB: Hyperledger Fabric Blockchain

$$D^{a,s} X_{a,s} \leq t^a \tag{15d}$$

$$e^{a,s} X_{a,s} \leq e^a \tag{15e}$$

$$R_b \geq R_a + \left(D^{a,s} + C^{a,s}\right) X_{a,s}, \qquad |I^{a,b} = 1 \tag{15f}$$

$$R_b - \sum_{s \in \mathcal{S}} D^{a,s} X_{a,s} - R_a \leq M\left(1 - \theta_{a,b}\right) | G^{a,b} = 0 \tag{15g}$$

$$R_b - \sum_{s \in \mathcal{S}} D^{a,s} X_{a,s} - R_a \geq -M\theta_{a,b} \qquad | G^{a,b} = 0 \tag{15h}$$

$$X_{a,s} + X_{b,s} + \theta_{a,b} + \theta_{b,a} \leq 3 \qquad | G^{a,b} = 0 \tag{15i}$$

$$\theta_{a,b} \in \{0, 1\} \tag{15j}$$

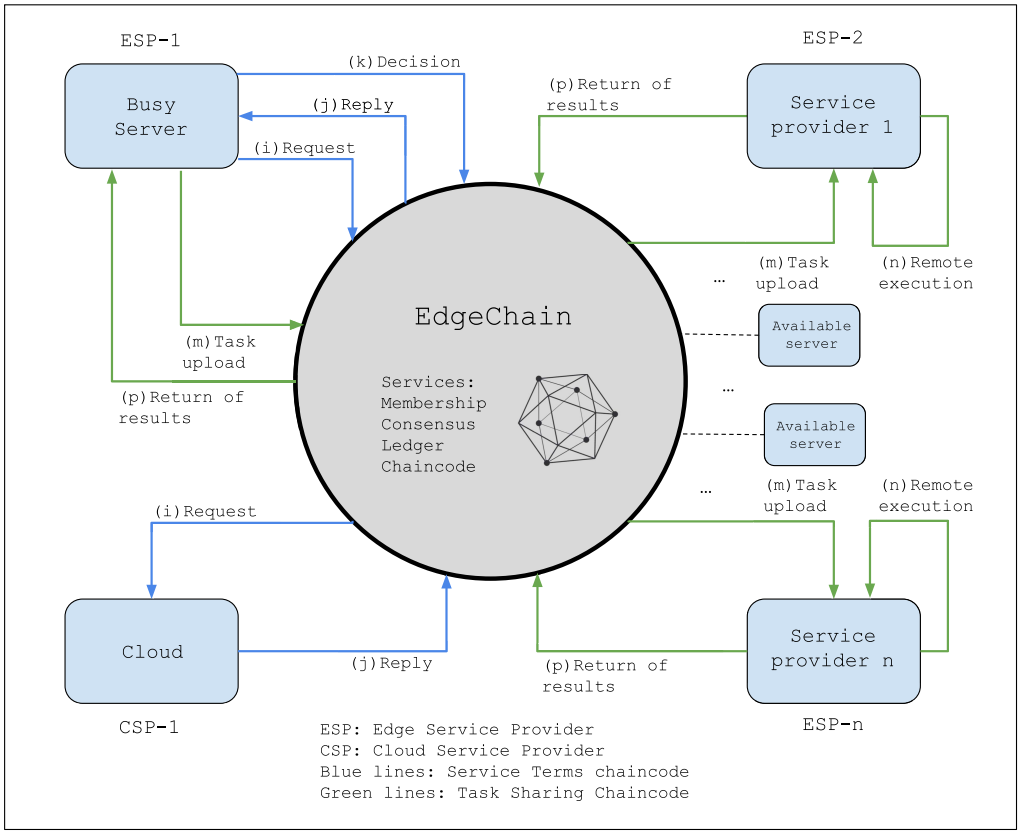

FIGURE 6. Blockchain players, their roles, and available transactions in *EdgeChain* network.

V. EDGECHAIN: A BLOCKCHAIN-BASED TASK SHARING SERVICE DEMO

A. PRELIMINARY ANALYSIS

EdgeChain is a blockchain-based solution designed for a consortium of ESPs to share computational tasks between organizations within a blockchain network. *EdgeChain* runs on the Hyperledger Fabric platform which is an open source permissioned blockchain with a trust model that consists of membership, consensus, ledger, and chaincode services so that server nodes in the network can engage in reliable, secure, and private task sharing services. ESPs are typically private tech companies (i.e., Google, Amazon, Microsoft, IBM) with computing infrastructure deployed in strategic locations near the end users they serve. ESPs might offer content delivery, Web services, and application services to users located at the bottom layer of a communications network. In our *EdgeChain* demo, five separate organizations, specifically, four ESP and one CSP, get into an infrastructure cluster to form a blockchain consortium to pool computer infrastructure and process computational tasks coming from stranded servers in the network. The whole premise of *EdgeChain* relies on the idea that a consortium of ESPs willing to share their idle resources would effectively reduce the latency associated with the execution of queued tasks in the system, and also increase the utilization efficiency of the computer infrastructure installed on the consortium.

There are four distinct roles in *EdgeChain*: (1) busy server, (2) cloud level, (3) available server, and (4) service provider. The interactions between busy servers and the cloud are overseen by the *Service Terms Chaincode* available on the global channel. More specifically, busy servers can request the task sharing service using the *(i) request* transaction. The cloud level, on the other hand, is equipped with a monitoring tool with an overview of the resources in the network and is in charge of replying to service requests with a set of candidate servers to take on the tasks. To do that, the cloud makes use of the *(j) reply* transaction. Finally, when busy servers make task sharing decisions to allocate queued computational tasks to a subset of available servers, they make use of the *(k) decision* to communicate their decisions to the global channel. After task sharing decisions have been reached, the interactions between busy servers and service providers are overseen by the *Task Sharing Chaincode*. In particular, the servers make use of the *(m) task upload*, *(n) remote execution*, and *(p) return of results* transactions to complete the lifecycle of the task sharing service on separate private channels. Figure 6 shows an illustration of the blockchain players, their roles, and the available transactions in the *EdgeChain* network. *EdgeChain* relies on the membership, consensus, ledger, and chaincode services provided by Hyperledger Fabric services to support the implementation of the task sharing solution. The deployment strategy presented in this

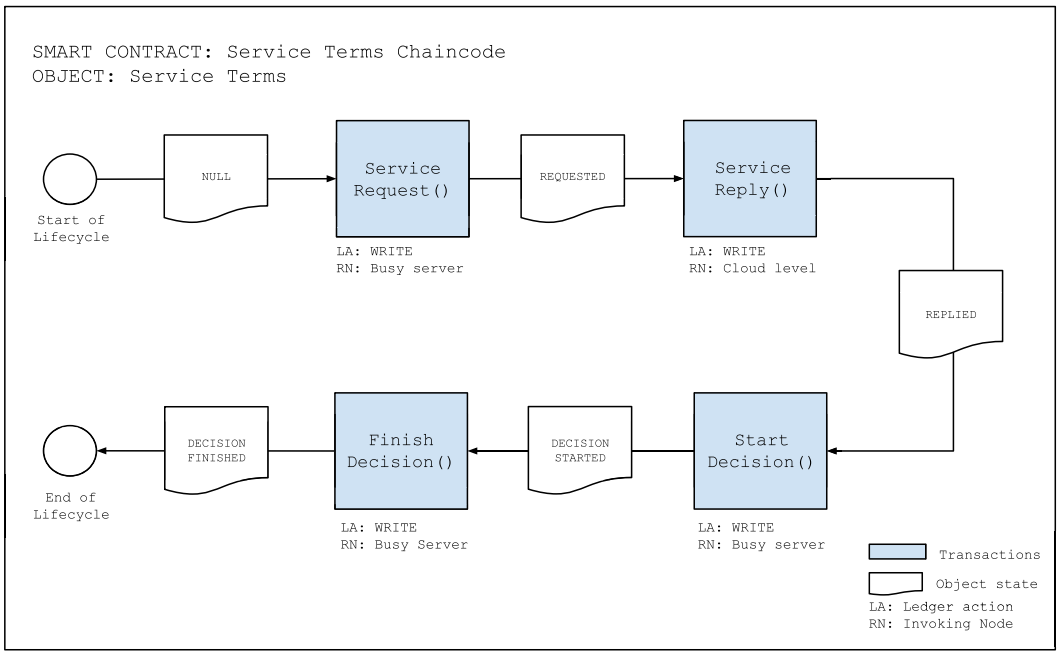

FIGURE 7. Lifecycle transitions of *Service Terms* objects.

chapter will reproduce the following sequence: (1) install the pre-requisites and components of the Fabric platform, (2) develop and deploy the chaincode with the logic of the task sharing solution, and (3) develop and install the necessary client applications that serve as the connection between the outside world and the blockchain network.

B. SMART CONTRACTS

EdgeChain participants (i.e., CSPs and ESPs) use chaincode available in the network to circulate computational tasks queued in busy servers. The blockchain model presented in Section III shows that the dynamics of the task sharing service can be described by the following sequence: (1) service request, (2) discovery of candidate nodes, (3) selection of service providers, and (4) task sharing contract. In this section, we present the details of the smart contracts that make up the core logic of the task sharing solution presented in this work.

1) SERVICE TERMS CHAINCODE

Service Terms Chaincode contains the logic of the first three stages of the task sharing model (i.e., service request, the discovery of candidate nodes, and selection of service providers) where the terms of the services are negotiated. We make use of Java programming language and Fabric Contract Shim API for coding the logic inside *Service Terms Chaincode*. Fabric Contract API provides a series of classes, methods, and interfaces to code contracts, transactions, and perform ledger operations. The smart contract is shared on the global channel and is accessible to the plenary of server nodes and the cloud level. In fact, busy servers and the cloud level might invoke transactions to negotiate computing resources in the network.

The objects of value in the global channel are named *Service Terms*. *Service Terms Chaincode* governs the process that handles *Service Terms* objects from initial creation to successful completion of their lifecycle. The transitions can be described by transactions that change the state of *Service Terms* objects circulating on the global channel. An illustration of the lifecycle of *Service Terms* objects is presented in Figure 7. From the lifecycle diagram, *Service Terms Chaincode* logic can be explained in terms of four transactions described as follows:

1) *Service Request:* Busy servers with queued computational tasks can initiate task sharing services with *serviceRequest()* transactions. When this function is invoked, busy servers hand over the strings *serviceID*, *serviceOwner*, *requestTime* and the JSON object *requestedResources* to put together transaction proposals for the network. If proposals are approved, new *Service Terms* objects are created and written to the channel ledger. The initial state of objects is set to *REQUESTED*.
2) *Service Reply:* The cloud level can reply to service requests with *serviceReply()* transactions. When this function is called, the cloud level hands over the strings *serviceID*, *serviceOwner* and the JSON object *availableResources* to create transaction proposals for the network. If transactions are approved, the *Service Terms* objects associated with provided *serviceID* parameters get updated on the channel ledger with the list of available resources in the network. Also, the state of the objects is moved from *REQUESTED* to *REPLIED*.

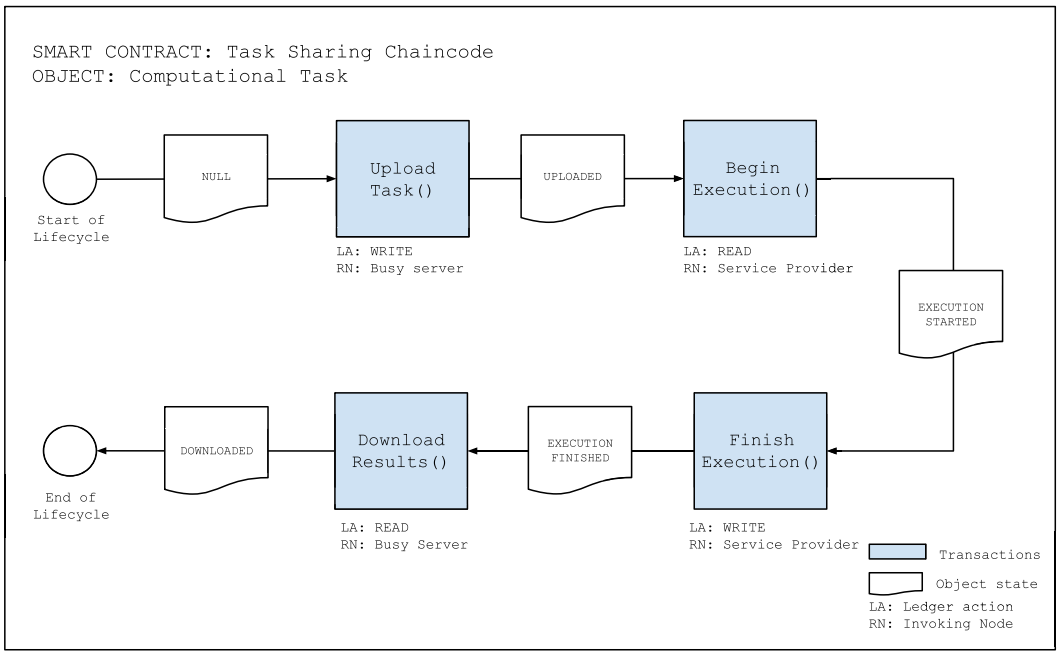

FIGURE 8. Lifecycle transitions of *Computational Task* objects.

3) *Start Decision:* When busy servers get notified that the cloud level has replied to their service requests, they must inform the network that they are ready to make task allocation decisions. To do that, they use *startDecision()* transactions. When this function is called, busy servers hand over the strings *serviceID* and *decisionStartTime* to construct transaction proposals for the network. After transactions are approved, *Service Terms* objects get updated on the ledger with timestamps that inform the network that busy servers have started to solve their task allocation problems. At this stage, the state of *Service Terms* objects is moved from *REPLIED* to *DECISION STARTED*.
4) *Finish Decision:* When busy servers have made their task allocation decisions, they use *finishDecision()* transactions to notify the blockchain network about this event. When this function is invoked, busy servers hand over the strings *serviceID*, *finishDecisionTime* and the JSON object *serviceproviders* to create transaction proposals and send them to the network. When transactions get approved, *Service Terms* objects get updated on the ledger with the set of servers that were selected to provide task sharing services. In other words, *finishDecision()* transactions culminate the negotiation process between busy servers and the cloud level. At this stage, *Service Terms* object states are moved from *DECISION STARTED* to *DECISION FINISHED* which also means the end to their lifecycle.

2) TASK SHARING CHAINCODE

Task Sharing Chaincode oversees the logic of the final stage of the task sharing model (i.e., the task sharing contract) where *Computational Tasks* objects are privately shared between busy servers and server nodes acting as service providers. As in *Service Terms Chaincode*, we use Java as the contract programming language and Fabric Contract API for the development of the logic inside *Task Sharing Chaincode*. This smart contract is shared on private channels and is accessible in pairs of busy servers and service providers.

The objects of value within the private channels are *Computational Tasks*. *Task Sharing Chaincode* controls the process that moves *Computational Task* objects from creation to successful completion of their lifecycle. Task transitions can be described by transactions that change the state of the tasks circulating on private channels. An illustration of the lifecycle of *Computational Task* objects is presented in Figure 8. From the lifecycle diagram, the logic in *Task Sharing Chaincode* can be fully explained with the four transactions described next:

1) *Upload Task:* Busy servers use *uploadTask()* transactions to start the sharing process with service providers over private channels. When this function is invoked, busy servers hand over the strings *taskID*, *uploadTime* and the JSON object *taskCode* to create a transaction proposal to the network. If proposals are approved, new *Computational Task* objects are created and written to the channel ledger. The initial state of task objects is set to *UPLOADED*.
2) *Begin Execution:* Service providers use *beginExecution* transactions to read *Computational Task* objects that were previously uploaded by busy servers to private channel ledgers. After tasks are read, they become available for remote execution. When this function is called, service providers hand over the

strings *taskID* and *execStartTime* to create transaction proposals for the network. If transactions are approved, the *Computational Task* objects associated with the provided *taskID* strings get updated with timestamps to notify the blockchain network that remote execution have started. Also, the state of the objects is moved from *UPLOADED* to *EXECUTION STARTED*.

3) *Finish Execution:* Service providers use *finishExecution* transactions to notify the blockchain that task execution have finished at their premises and also upload task results to the channel ledger. When this function is called, service providers hand over the strings *taskID*, *execFinishTime* and the JSON object *taskResults* to create transaction proposals for the network. If transactions are approved, *Computational Task* objects get updated on the ledger with timestamps that inform the network that remote execution have finished. At this point, the state of *Computational Task* objects is changed from *EXECUTION STARTED* to *EXECUTION FINISHED*.
4) *Download Task Results:* When busy servers get notified that remote execution of tasks have finished, they use *downloadResults()* transactions to download task results from the channel ledger. When this function is invoked, busy servers hand over strings *taskID* and *downloadTime* to create transaction proposals for the network. If proposals are approved, *Computational Task* objects get updated on the ledger with timestamps that inform the network that task results have been downloaded. Also, *downloadResults()* transactions end the lifecycle *Computational Task* objects and move their states from *EXECUTION FINISHED* to *DOWNLOADED*.

C. CLIENT APPLICATIONS

Client applications can interact with the *EdgeChain* network to submit transactions that may force changes on objects stored in the blockchain. *EdgeChain* clients will connect to the blockchain using a Fabric Java SDK (a set of connection functions) and gateway peers inside their parent organizations to reach other nodes in the network. The configuration of gateway peers may be found in the connection profile generated by parent organizations before joining the network consortium. They make use of their local copy of the chaincode installed in the blockchain channel and the Fabric Contract API to assemble transaction proposals that will be submitted to the network.

The gateways provide an entry point for clients to access channels and chaincode in the blockchain network. To do that, clients first retrieve *x.509* digital certificates from their wallets corresponding to pre-authorized users with clearance to access the network. Then, they load the connection profile with designated gateway peers for their parent ESP organizations. Both, identity wallets and connection profiles, are available on the local file systems of the physical nodes they run on. At this stage, clients may call the functionality inside the chaincode to prepare transaction proposals and set off the Fabric consensus mechanism. Client applications are a key part of the *EdgeChain* solution due to their ability to hook up the outside world to the blockchain infrastructure. An illustration of the sequence of events executed in the network for proper interaction between clients and the blockchian is presented in Figure 9.

Depending on their physical location in the network, client applications may monitor the performance of edge servers, manage queued tasks, and solve task allocation problems. More specifically, ESPs the CSP in *EdgeChain* can invoke transactions available in *Service Terms Chaincode* and *Task Sharing Chaincode* to circulate computational tasks in the edge network. This section presents the details of the three client applications developed for *EdgeChain*: (1) *BusyServerApp*, (2) *CloudLevelApp*, and (3) *PrivateSharingApp*.

1) BUSYSERVERAPP

BusyServerApp clients are Java programs installed on the ESP servers that monitor incoming application requests (i.e., computational jobs) from mobile users and request computer resources on their behalf. An instance of *BusyServerApp* may call the *Service Terms Chaincode* installed in the global channel to start the negotiation of task sharing services with the blockchain network when the server nodes have queued tasks that need processing services. *BusyServerApp* may then submit transactions on behalf of stranded servers looking to get resources from the edge network. The clients are, in fact, the middle man between busy servers and the blockchain network for the negotiation process of task sharing services. In particular, they can request computing resources, solve task allocation problems, and notify the blockchain about allocation decisions. A summary of the labor inside *BusyServerApp* clients is presented next.

1) Retrieve *x.509* client identities from their local ESP wallets
2) Connect to designated ESP gateway peers using the information in the connection profiles of their parent organizations
3) Access the global channel using the Java SDK functionality
4) Connect to *Service Terms Chaincode* and submit transaction proposals to the network
5) Stay alert of transaction notifications and process responses accordingly

With respect to the decision engine of our model, *BusyServerApp* clients integrate the Google optimization tool (*Google OR-Tools*) [67] to solve the task scheduling problem designed for the *EdgeChain* solution (the problem is explained in section 3.E *"Task Sharing Model at the Server Level"*). *Google OR-Tools* is an open source optimization software available for multiple programming languages including Java that offers a variety of solvers for combinatorial optimization. *Google OR-Tools* specializes in finding optimal schedules for allocation problems composed of a set of task objects that

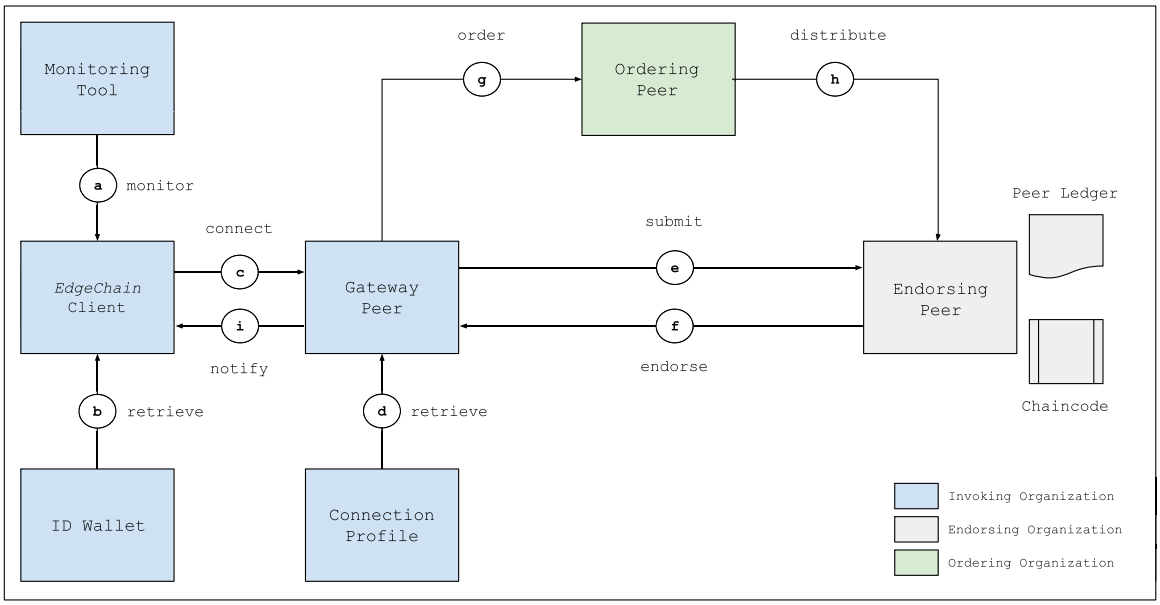

FIGURE 9. Communication sequence between client applications and the blockchain inside the *EdgeChain* network.

need to be distributed among a fixed set of resources that provide labor. *EdgeChain* makes use of the GLOP solver to find optimal solutions for the linear program in (15a).

2) CLOUDLEVELAPP

CloudLevelApp clients are Java programs installed on the cloud datacenter (i.e., CSP) to negotiate task sharing service requests with busy servers. The client calls functions inside *Service Terms Chaincode* to reply to service requests and notify the network with available computer resources in *EdgeChain* network. An instance of the *CloudLevelApp* may submit transactions on behalf of the monitoring module sitting at the cloud level. The module has an overview of the computer resources in the network and can shortlist a set of candidate servers to take on computational tasks queued in the system. From the gateway peer location, the *CloudLevelApp* client may call *serviceReply* transactions to update the blockchain ledger with the information about available computer resources in the edge network. A summary of the labor inside *CloudLevelApp* client is presented next:

1) Retrieve *x.509* client identities from the local CSP wallet
2) Connect to the designated CSP gateway peer using the information in the connection profile of the parent organization
3) Access the global channel using Java SDK functionality
4) Connect to *Service Terms Chaincode* and submit transaction proposals to the network
5) Stay alert of transaction notifications and process responses accordingly

For our demo, we assume that all the servers in *EdgeChain* are potential candidates to accept queued tasks in the network. However, in production scenarios, there is a variety of open-source applications available that might be installed on the cloud datacenter to monitor computer resources and blockchain performance in the edge network. This information may be used as input for the allocation decision engine of our model.

3) PRIVATESHARINGAPP

PrivateSharingApp clients are Java programs installed on ESP servers to oversee the task sharing process between busy servers and service providers. They may call the functionality available in the *Task Sharing Chaincode* to upload/download data of queued tasks and their results. An instance of the *PrivateSharingApp* can submit transactions on behalf of busy servers and service providers to privately manage the task sharing process between each other. In fact, they can upload, remote execute, and download the results of queued computational tasks in the network. *PrivateSharingApp* clients have two versions, one for busy servers and one for service providers that get used depending on the server's role in the task sharing service. A summary of the labor inside *PrivateSharingApp* clients is presented next:

1) Retrieve *x.509* client identities from local ESP wallets
2) Connect to designated ESP gateway peers using the information in the connection profiles of the ESP organizations
3) Access private channels using Java SDK functionality
4) Connect to *Task Sharing Chaincode* and submit transaction proposals to the network
5) Stay alert of transaction notifications and process responses accordingly

D. NETWORK SETUP

The first step towards launching a Fabric network consists in the installation of a set of software pre-requisites on the

nodes partaking in the network. Hyperledger Fabric is a container-based platform that uses container management tools, programming languages, packet managers, and binaries to facilitate the operation of the blockchain network. Fabric binaries, in particular, are the platform-specific machine language executable files to set up and run the Fabric blockchain. Once the required software is installed on the blockchain nodes, the Fabric network is ready to be launched. For the deployment of *EdgeChain*, we have four PCs to simulate the infrastructure of the organizations taking part in the blockchain consortium. Every computer will host a set of docker containers with the assigned Fabric components defined in the Fabric configuration files. The containers include CAs, ordering service peers, organization peers, peer databases, CLIs, chaincode packages, and client application packages. The blockchain configuration files and smart contract code that make up the *EdgeChain* network are available in the *angelovera/TaskSharingServiceDemo* Github repository [69].

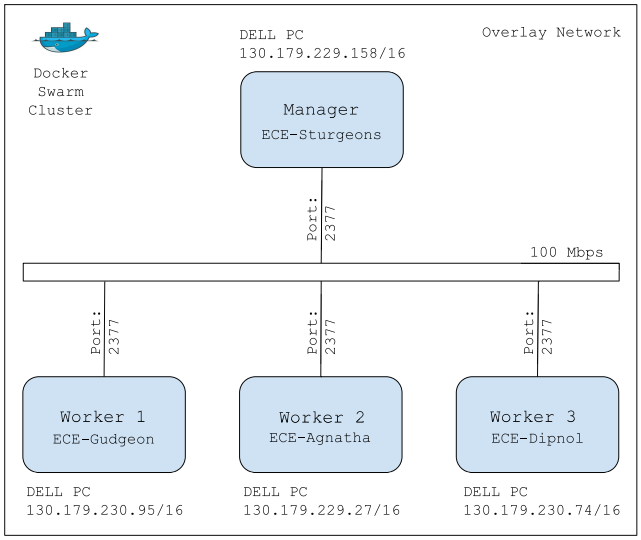

FIGURE 10. Illustration of the *EdgeChain* docker swarm cluster.

E. PHYSICAL CONNECTIONS

The four *EdgeChain* PCs are physically connected to a hard-wired Local Area Network with a data-rate capacity of 100 Mbps. The computers are logically organized in a cluster with the help of Docker Swarm orchestration software. Docker-swarm powers the implementation of container-based networks across multiple physical or virtual hosts. This is a deliberate design choice so that the set of available PCs can be managed from a single administration point. In the case of *EdgeChain*, the cluster is controlled by one node that acts as cluster manager and another three nodes that act as followers or workers. The manager node is the single point of administration of the physical network, therefore, all the Fabric components and their docker containers can be deployed from this location. An illustration of the *EdgeChain* docker swarm cluster can be found in Figure 10.

F. NETWORK COMPONENTS

Hyperledger Fabric is a container-based blockchain that uses isolated software units (i.e., software containers) to deploy network components on the physical infrastructure. *EdgeChain* is a Fabric-based network that recreates a distributed version of a cloud/edge architecture. More specifically, four ESPs and one CSP get together to form a blockchain consortium to launch a decentralized task sharing service that delivers computing services to software applications located behind the edge infrastructure. The consortium accommodates five organizations (i.e., ESPs and CSP) across the four computers available for *EdgeChain*. Every organization is composed of one CA in charge of generating cryptographic identities inside the organization, one organization peer that hosts the chaincode with the task sharing logic, one instance of a Couch Database (CouchDB) that keeps a copy of the world state and blockchain, and a Command Line Interface (CLI) to interact with the peers. In addition, *EdgeChain* implements ordering peers for the ordering service, and the membership service to validate identities within the network. An illustration of the arrangement of docker containers inside *EdgeChain* participating organizations (i.e., CSP and ESPs) can be found in Figure 11.

Fabric provides a CA tool called *Fabric-CA* that reproduces a PKI to create digital identities in the network. *Fabric-CA* generates cryptographic identities through a process called *enrollment* that is configurable to fit the amount of crypto material required by the network. *Fabric-CA* might be used during development stages, however, when moved to production, it should be replaced with commercially available CAs that are trusted by the Internet industry [68]. Following the spin-up of containers with the CAs, *Fabric-CA* tool generates cryptographic identities for network participants (i.e., organizations, peers, client users, etc). They use the crypto identities to sign their attempts to interact with the blockchain network. After crypto material generation, the rest of Fabric components are brought up to complete the formation of the network. The spin-up of network containers is initiated by the manager node in the docker swarm cluster using docker commands. At this point, ordering peers, organization peers, CouchDB instances, and CLIs should be deployed across network nodes. The container profiles are available in *YAML* configuration files that are used by docker commands during the spin-up process.

Following the creation of Fabric components, *EdgeChain* consortium is ready to create application channels. To do that, Fabric provides the *ConfigTxGen* tool that allows the creation of channel configuration blocks and related channel artifacts (i.e., channel update transaction and anchor peers information). For channel creation, the *ConfigTxGen* tool receives the configuration file of the network as an input. The network contains one global channel with the plenary of edge server nodes, and $\binom{n}{2}$ private channels (n is the number of server nodes) to privately connect every combination of two servers in the network. A summary of

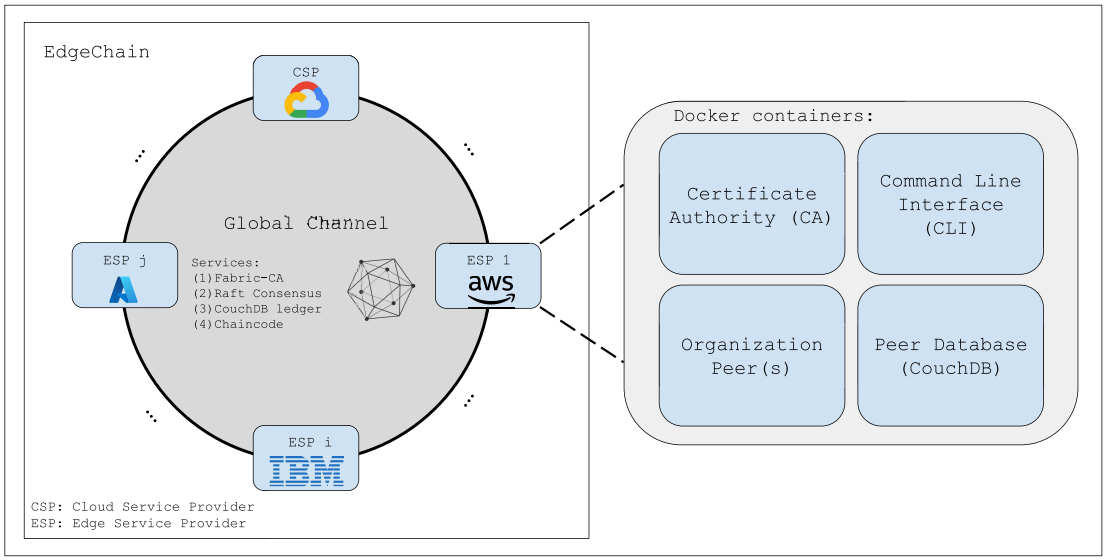

FIGURE 11. Arrangement of docker containers inside participating organizations in the *EdgeChain* network.

the components of the *EdgeChain* network can be found in Table 7.

VI. PERFORMANCE EVALUATION RESULTS

The *EdgeChain* network runs on a physical cluster of four identical computers, the network consists of a Fabric-CA PKI instance, a Raft-based Ordering Service module, a consortium of five organizations, seven blockchain channels, two pieces of chaincode, and three client applications that make up the blockchain-based task sharing solution. Every computer has a 3.40 GHz x 8 i7 core processor, 7.16 GiB of available memory, and 500 GBs of storage. In this section, we evaluate the impact of a few configurable parameters of the Fabric platform on the transaction latency and transaction throughput induced by the blockchain network. Latency is defined as the time (in seconds) taken by transactions from when they leave client applications to when they get committed to the blockchain ledger. Throughput, on the other hand, is defined as the rate (in transactions per second) at which transactions are committed to the blockchain ledger. The evaluation of transaction latency and transaction throughput is the standard practice to measure the response time and scalability of a blockchain solution. We use these two quantities as the primary metrics to analyze the performance of *EdgeChain*. If not otherwise specified, the results presented in this section are averaged over multiple runs of the same experiment. The configuration parameters for the experiments presented in subsections *A*, *B*, and *C* can be found in Table 8.

A. IMPACT OF STATE DATABASE

Let us recall that Hyperledger Fabric currently offers two ledger implementations to store KV data related to blockchain objects: (1) LevelDB and (2) CouchDB. LevelDB ledger is the default implementation of a state database in Fabric nodes that comes embedded inside core peers. It is lightweight but supports only a limited number of simple operations on the database. CouchDB ledger, on the other hand, is an SQL database that runs separately from peer nodes. It supports advanced operations on the database, complex indexing methods, and complex data structures. For this experiment, all the Fabric configuration parameters are fixed except for the ledger state database. Figures 12 and 13 show the impact of the type of ledger database on the average transaction latency experienced by transactions. *Service Terms Chaincode* and *Task Sharing Chaincode* transactions are taken into account. The number of transaction runs is 5000.

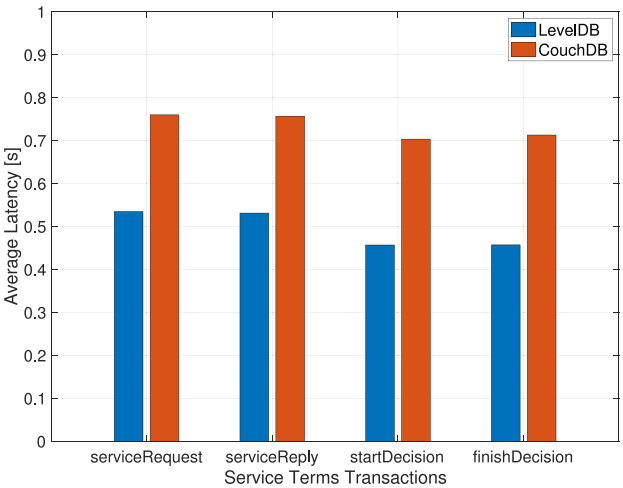

FIGURE 12. Average latency of *Service Terms Chaincode* transactions as a function of state database.

1) OBSERVATION 1

The results show that the latency experienced by transactions for the CouchDB ledger implementation is significantly greater when compared to the latency experienced when

TABLE 7. Hyperledger fabric components of the *EdgeChain* network.

Component	Container Name	Physical Node	Network Alias
System:			
Orderer CA	ca.orderer	Manager	ca.orderer.edgechain.com:10054
Orderer Peer	orderer	Manager	orderer.edgechain.com:7050
ESP1:			
CA	ca.org1	Worker1	ca.org1.edgechain.com:7054
Peer	peer0.org1	Worker1	peer0.org1.edgechain.com:7051
Peer DB	couch.peer0.org1	Worker1	couch.peer0.org1.edgechain.com:5984
CLI	cli.org1	Worker1	cli.org1.edgechain.com
ESP2:			
CA	ca.org2	Worker2	ca.org2.edgechain.com:8054
Peer	peer0.org2	Worker2	peer0.org2.edgechain.com:9051
Peer DB	couch.peer0.org2	Worker2	couch.peer0.org2.edgechain.com:7984
CLI	cli.org2	Worker2	cli.org2.edgechain.com
ESP3:			
CA	ca.org3	Worker3	ca.org3.edgechain.com:9054
Peer	peer0.org3	Worker3	peer0.org3.edgechain.com:11051
Peer DB	couch.peer0.org3	Worker3	couch.peer0.org3.edgechain.com:9984
CLI	cli.org3	Worker3	cli.org3.edgechain.com
ESP4:			
CA	ca.org4	Manager	ca.org4.edgechain.com:11054
Peer	peer0.org4	Manager	peer0.org4.edgechain.com:13051
Peer DB	couch.peer0.org4	Manager	couch.peer0.org4.edgechain.com:11984
CLI	cli.org4	Manager	cli.org4.edgechain.com
CSP:			
CA	ca.org5	Manager	ca.org5.edgechain.com:12054
Peer	peer0.org5	Manager	peer0.org5.edgechain.com:15051
Peer DB	couch.peer0.org5	Manager	couch.peer0.org5.edgechain.com:13984
CLI	cli.org5	Manager	cli.org5.edgechain.com

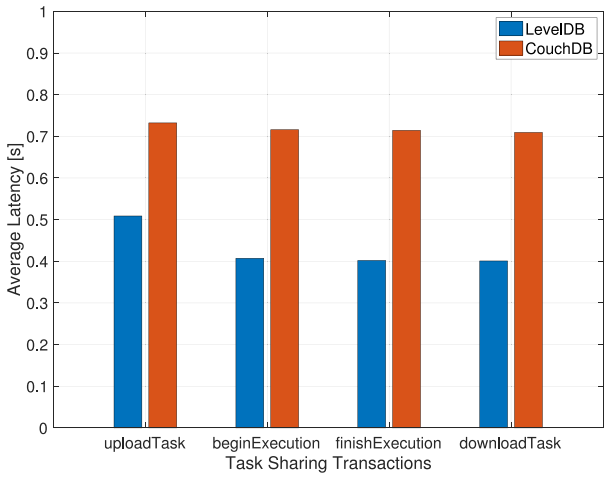

FIGURE 13. Average latency of *Task Sharing Chaincode* transactions as a function of state database.

LevelDB ledger is installed. The main reason for the difference is that LevelDB is a state database embedded in the peer nodes while CouchDB is a database that runs separately from peer nodes using REST APIs over a secure HTTP connection. The communication between peer nodes and the REST APIs adds additional steps to the write/read operations that transactions force on the CouchDB ledger. Across transactions, there is also a slight difference in the latencies introduced by the two pieces of chaincode. As for the *Service Terms Chaincode*, the complexity of *ServiceRequest* and *ServiceReply* transactions is higher than that of *startDecision* and *finishDecision*. The higher number of KV ledger operations introduced by the first two has a slight impact on the latency being measured. This effect can be seen in both, CouchDB and LevelDB, implementations. As for the *Task Sharing Chaincode*, the effect of KV ledger operations can also be seen in the latency introduced by *uploadTask* transaction with respect to the other three (i.e., *beginExecution*, *finishExecution*, and *downloadTask*) although the effect is more noticeable in the LevelDB implementation.

2) GUIDELINE

LevelDB ledger is a simpler and faster option than CouchDB ledger and might be a better design choice for latency-sensitive applications that manage data with low complexity structures. CouchDB, on the other hand, is slower but can handle more

TABLE 8. Blockchain parameters for the experiments in subsections A, B, and C.

	Database Experiment (A)	Endorsement Experiment (B)	Tx Arrival Rate Experiment (C)
State Database	LevelDB and CouchDB	LevelDB	LevelDB
Endorsement policy	Majority (default)	NOutOf, N=1,3,5	NOutOf, N=1
Tx arrival rate	1 tx/s	1 tx/s	[1,13] tx/s
Block size	1 tx/block	1 tx/block	{1,3,5,7} tx/block
Block timeout	1 s	1 s	1 s

sophisticated data structures. It may be a good choice for applications that can tolerate higher response latencies.

3) ACTION

Move on to the next experiments setting LevelDB as the state database for *EdgeChain* network.

B. IMPACT OF ENDORSEMENT POLICY

Let us recall that, at the peer level, the consensus mechanism in Fabric networks rely on the idea of endorsements. Endorsement policies are the rules that define the minimum number of digital signatures that must be collected from the blockchain network before transaction proposals get approved and moved to the ordering service. When client peers submit transaction proposals to a blockchain channel, they are sent to endorsing peers defined on the channel. Endorsing peers receive the proposals and simulate them with their local copy of the chaincode. If the simulation process produces the same output as the proposals, transactions are signed with the endorser's private key and are sent back to the client that created them. Endorsing signatures are a method to validate that transaction results strictly follow the installed logic on the blockchain. From a macro perspective, endorsement policies dictate the level of democratization for the voting system to approve transactions, that is, minority, majority, or unanimity approvals. The policies are the result of the combination of three logic operators: (1) *AND*, (2) *OR*, and (3) *NOutOf*. The operands are always members of the organizations participating in the blockchain consortium. For example, an endorsement policy defined by the expression $AND(Org1MSP.peer, Org2MSP.peer)$ means that any peer in both, Org1 and Org2, must sign a transaction proposal before it is declared valid. The expression $NOutOf(1, Org1MSP.peer, Org2MSP.peer, Org3.peer)$, on the other hand, means that at least 1 out of the 3 organizations in the policy (i.e., Org1, Org2, and Org3) must endorse a transaction proposal to get it approved. For this experiment, all the Fabric configuration parameters are fixed except for channel endorsement policies. Figures 14 and 15 show the impact

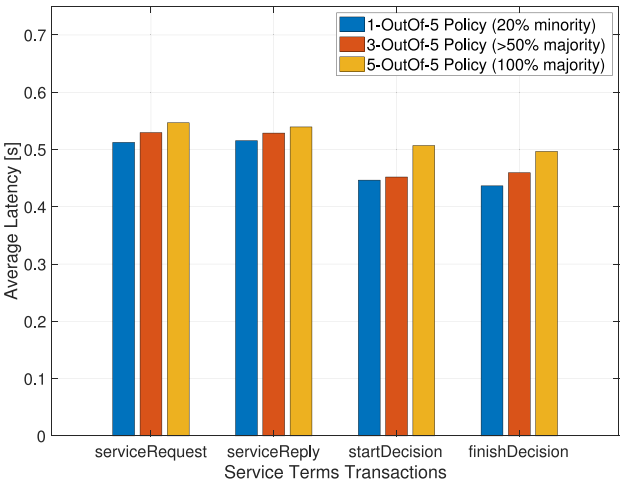

FIGURE 14. Average latency of *Service Terms Chaincode* transactions as a function of endorsement policy.

of endorsement policies in the average transaction latency experienced by transactions. *Service Terms Chaincode* and *Task Sharing Chaincode* transactions are taken into account. The number of transaction runs is 5000.

1) OBSERVATION 2

For the *Service Terms Chaincode*, we evaluate three endorsement policies: (1) minority approval (20% endorsement signatures, 1 out of 5 organizations), (2) simple majority approval (≥50% endorsement signatures, 3 out of 5 organizations), and (3) unanimity approval (100% endorsement signatures, 5 out of 5 organizations). For the *Task Sharing Chaincode*, we evaluate two endorsement policies: (1) weak majority approval (50% endorsement signatures, 1 out of two organizations), and (2) unanimity approval (100% endorsement signatures, 2 out of 2 organizations). The results show that the latency experienced by transactions is indeed affected by the number of signatures required by the endorsement policy. As for the *Service Terms Chaincode*, the minority approval and simple majority approval show similar latency times. The unanimity approval, on the other

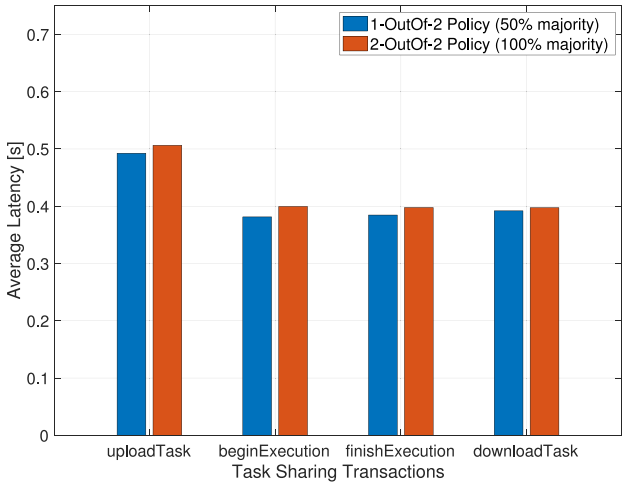

FIGURE 15. Average latency of *Task Sharing Chaincode* transactions as a function of endorsement policy.

hand, slightly increases the latency compared to the other two policies. As for the *Task Sharing Chaincode*, the unanimity approval forces a slight increment in the latency for the weak majority approval, however, the increment is almost negligible. In general, the increment in transaction latency can be explained in terms of three main operations that take place during the endorsement validation phase. First, *x.509* identity certificates of endorsing peers are de-serialized from the membership module back to the proposing peer. Second, once de-serialized, the certificates are validated against the list of organization MSP identifiers. And third, the signatures of endorsed transactions have to be verified to confirm the identity of endorsing peers. The three operations are executed by the proposing peers every time endorsed transactions enter their premises. We can see that the effect of different endorsement policies is mostly experienced by the *Service Terms Chaincode* which operates in the *global channel*. The *global channel* contains 5 organizations and an equal number of peers, therefore, a change in the number of required signatures to approve transactions in this channel has greater latency impact than similar changes in channels with less number of participants. This is true simply by the lesser number of operations that must take place in smaller channels during the endorsement validation phase. This can be corroborated with the behavior of transactions in the *Task Sharing Chaincode* which operates in private channels with only two organizations and two peers.

2) GUIDELINE

The complexity of endorsement policies is a factor that impacts the scalability of the blockchain network. When a network requires a larger number of endorsements, transactions must be simulated and signed at a higher number of peer nodes during the endorsement phase. Therefore, the higher the number of required signatures, the higher the latency experienced by transactions. For better performance, set endorsement policies with only a few required signatures.

3) ACTION

Move on to the next experiments setting the endorsement policy to one required signature (*NOutOf*, $N = 1$) for all *EdgeChain* smart contracts.

C. IMPACT OF TRANSACTION ARRIVAL RATE AND BLOCK SIZE

In a Fabric network, transactions are ordered and packed into blocks at the ordering service. Once created, blocks are sent back to organization peers for final verification and committing to their local copies of the blockchain ledger. The block size is a configurable parameter of the blockchain network that defines the number of packed transactions in a block. Unlike the endorsement phase in which processing is done in a per-transaction style, the processing of blocks at the peer level is done one block at a time. In fact, there is a tight relationship between block size, transaction latency, and transaction throughput inside the network. A deficient pick of block size may impact the speed at which blocks are created when the arrival rate is not sufficiently. This may have transactions wait too long at the ordering service causing bottlenecks in the network. For a better picture of this latency and throughput experiment, we study these parameters in conjunction with the transaction arrival rate. For this experiment, all the Fabric configuration parameters are fixed except for the block size. Figures 16 to 19 show the impact of block size on the average transaction latency and the average transaction throughput experienced by *EdgeChain*. The network is tested under multiple block sizes and transaction arrival rates.

1) OBSERVATION 3

We evaluate four block size policies (blocksize $= 1, 3, 5, 7$) at different transaction arrival rates to catch potential bottlenecks in the network. The tension between latency and throughput becomes clear when the block size policy and the transaction arrival rate vary. For blocksize $= 1$, the throughput increases linearly as expected before reaching the saturation point. When the arrival rate gets close to the saturation point, the latency increases significantly from less than half of a second to a few seconds. This effect can be explained by the growing number of service transactions waiting at the ordering service queue when the arrival rate increases. The waiting time at the queue affects the ultimate commit latency time and becomes a bottleneck in the system. The behavior can be seen in Figure 16.

2) OBSERVATION 4

When we increase the block size to 3, 5, and 7, the transaction latency remains unaltered at approximately 1.3 seconds when the arrival rate is below the block size. This can be explained due to the block generation time being conditioned by the *block timeout* parameter that gains prominence when the rate of incoming transactions is below the size of a block. In this scenario, ordered transactions have to wait

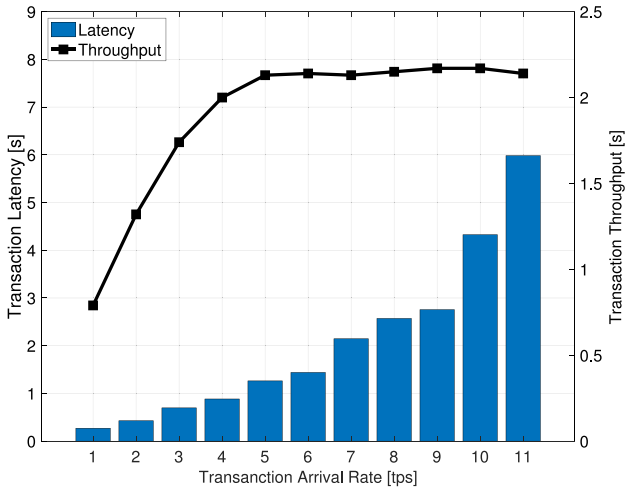

FIGURE 16. Average transaction latency and average transaction throughput versus transaction arrival rate with block size equal to 1.

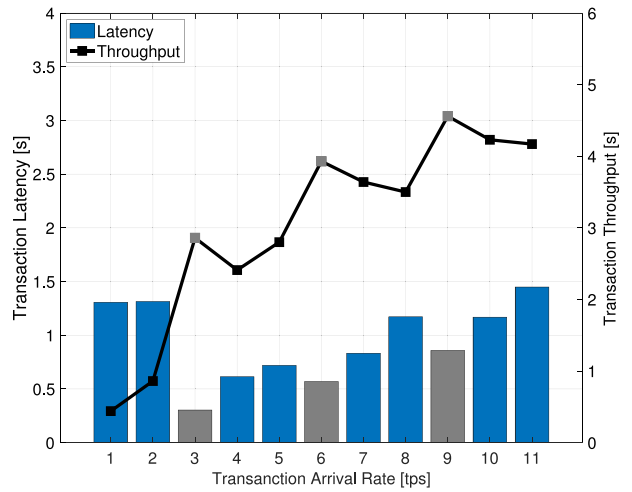

FIGURE 17. Average transaction latency and average transaction throughput versus transaction arrival rate with block size equal to 3.

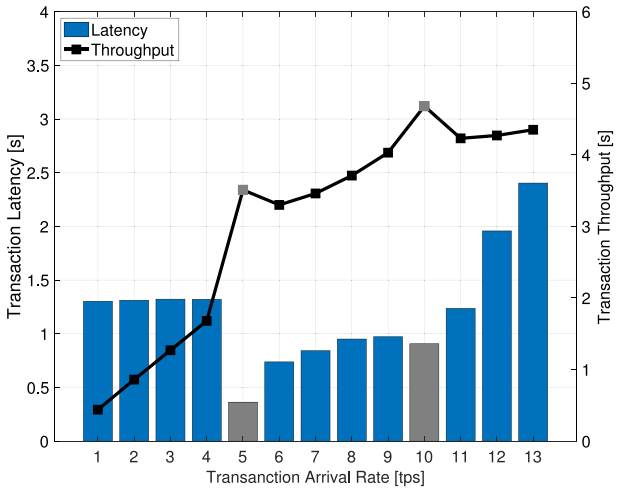

FIGURE 18. Average transaction latency and average transaction throughput versus transaction arrival rate with block size equal to 5.

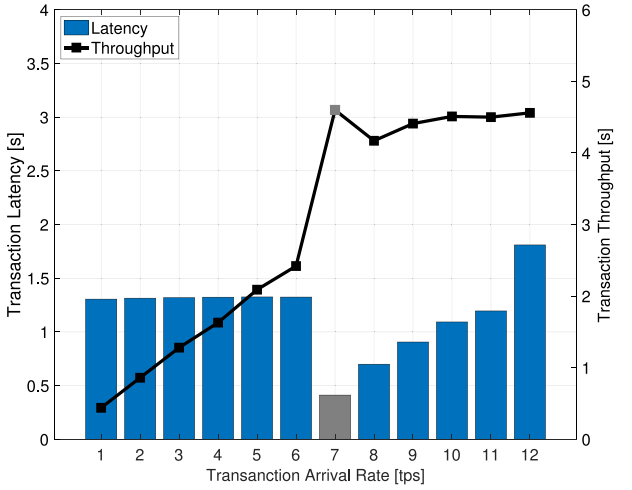

FIGURE 19. Average transaction latency and average transaction throughput versus transaction arrival rate with block size equal to 7.

for at least *"block timeout"* seconds before getting committed to the network. In our tests, the *block timeout* is set to 1 second. For arrival rates above the block size, the transaction latency increases with the arrival rate. An interesting behavior is spotted at the points where the arrival rate matches the block size or its multipliers ($blocksize = N$, $multipliers = 2*N, 3*N, \ldots$). At these points, the network experiences a decrease in transaction latency that can be explained by a faster generation of blocks given that the number of incoming transactions in a batch meets the block transaction quota exactly hence no remaining transactions have to wait in the ordering service queue for the next round of block generation and commit phase. On average, this effect translates in a slight decrease in committing times that can be spotted on the grey bars in Figures 17, 18, and 19.

3) OBSERVATION 5

Same as transaction latency, the transaction throughput increases as we increase the block size ($blocksize = 3, 5, 7$). Again, for arrival rates equal to the block size or its multipliers, the transaction throughput experiments a boost that can be explained in similar terms as the latency case. At these points, blocks are created at a faster rate because of the synchronization between the transaction arrival rate and the transaction quota at the block level that, on average, reduces the waiting time of service transactions at the ordering service hence increasing the throughput produced by the blockchain. The effect is highlighted with grey square markers in Figures 17, 18, and 19.

4) GUIDELINE

To achieve higher transaction throughput and lower transaction latency, always match the block size to the transaction arrival rate when the network sits below the saturation point. We recognize that when the blockchain finds optimal latency and throughput levels, the network may offer faster and more scalable task sharing services.

VII. CONCLUSION

We have proposed a conceptual framework for task sharing collaboration in MEC architectures using the Hyperledger Fabric platform. MEC networks can benefit from the notarization, ownership, and chain-of-custody services offered by the Hyperledger Fabric blockchain to address security and privacy concerns related to collaboration mechanisms for edge computing servers. The proposed MEC framework supports decentralized computing services with dynamic task scheduling and unified management of resources while maintaining security capabilities with a special focus on the integrity and confidentiality of tasks and their data

At the blockchain level, we have proposed a task sharing design that accommodates a set of edge servers and the cloud level that get together to form a consortium in the Hyperledger Fabric platform. The dynamics of the solution can be described in four phases: (1) service request, (2) discovery of candidate nodes, (3) selection of service providers, and (4) task sharing contract. The model offers enhanced security features that leverage the permissioned nature of Fabric to validate identities and allowed behavior within the network. The model also offers enhanced privacy features powered by a smart contract design that makes use of multiple decoupled Fabric channels with separate operation rules, ledgers, and peer-to-peer communication networks. This design choice guarantees that the computational tasks circulating in the network are only exposed to servers participating in task sharing services. At the server level, the task sharing scheme is modeled as a multi-processor task scheduling problem that allocates a set of precedent-dependent computational tasks among a set of available servers by jointly optimizing the aggregated utility and the makespan of the tasks. In summary, we introduce a reference task-sharing model at the blockchain and server levels that establish a block-based agreement among multiple ESPs that enables macro deployment and cross-collaboration among them.

We also have presented the implementation details of *EdgeChain*, a proof of concept demo of the task-sharing model. *EdgeChain* is a Hyperledger Fabric-based solution designed for a consortium of ESPs to share computational tasks between organizations within a blockchain network. *EdgeChain* network runs on a physical cluster of four computers and consists of the combination of Fabric-CA PKI, a Raft-based Ordering Service, a consortium of five organizations (four ESPs and one CSP), seven blockchain channels, two smart contracts, and three client applications that make up the task sharing solution. The implementation of *EdgeChain* shows that our task sharing model is feasible, however, there is still much room for improvement to reduce the latency introduced by service transactions, increase the throughput produced by the blockchain, and improve the overall features offered by the task sharing model.

Our results show that the latency overhead introduced by task sharing transactions can be minimized by optimizing the configurable parameters of Fabric (i.e., endorsement policy, state database, block size, batch timeout, etc). Also, it is possible to introduce changes to the task sharing model at the blockchain level to minimize the number of required transactions. A priori, the transactions in our model could be re-organized to reduce the number of operations on the ledger. Recall that transactions in a blockchain network ultimately translate into KV write/read operations on the ledger which add up to the latency overhead introduced by the system, write operations in particular.

Sharing computational tasks can also be seen as a trading problem rather than a multi-processor scheduling problem. Our current approach does not bring market tensions to the picture although we believe that market models such as auctions and their variations might capture the nuances of the sharing process more adequately. Recall that the ESPs are normally private companies with profit-driven economic incentives. In that case, a viable collaboration model for a consortium of ESPs should include a pricing model that captures the economic reward for the providers of sharing services. A market model can seemingly introduce a pricing mechanism for the payments and it can be easily programmed as a cryptocurrency in the logic of the Hyperledger Fabric blockchain.

Finally, the mobility of end users is an issue that our model does not address. Service break-offs may happen if mobile users move across coverage areas of different edge servers. In that scenario, the provision of smooth task ownership handover between server nodes is one of the challenges for the evolution of our task sharing model. Also, the computational jobs in our model follows the acyclic dependency restriction. However, that assumption may not necessarily coincide with the structure and complexity of real end-user applications.

We believe that any future evolution of our model should focus on two tasks: (1) reduce the latency introduced by the blockchain network, and (2) increase the overall features offered by the task sharing framework.

REFERENCES

[1] B. Cao, L. Zhang, M. Peng, and M. A. Imran, *Wireless Blockchain: Principles, Technologies and Applications*. Chichester, U.K.: Wiley-IEEE Press, 2022, ch. 5.

[2] G. Fridgen, S. Radszuwill, N. Urbach, and L. Utz, "Cross-organizational workflow management using blockchain technology—Towards applicability, auditability, and automation," in *Proc. 51st Hawaii Int. Conf. Syst. Sci.*, 2018, pp. 1–10, doi: 10.24251/HICSS.2018.444.

[3] F. Milani, L. García-Bañuelos, and M. Dumas. "Blockchain and business process improvement." BPTrends Newsletter. Oct. 2016. [Online]. Available: https://www.bptrends.com/bpt/wp-content/uploads/10-04-2016-ART-Blockchain-and-Bus-Proc-Improvement-Milani-Garcia-Banuelos-Dumas.pdf

[4] A. Gupta and R. K. Jha, "A survey of 5G network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015, doi: 10.1109/ACCESS.2015.2461602.

[5] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update*, vol. 19, Cisco, San Jose, CA, USA, 2017.

[6] M. Gantz and P. Miller, *The Salesforce Economy: Enabling 1.9 Million New Jobs and 389 Billion Dollars in New Revenue Over the Next Five Years*, vol. 19, Int. Data Corp., Needham, MA, USA, 2016.

[7] Y. Mao, C. You. J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2019.

[8] W. Johnston, K. Sparks, and B. Daly, "5G edge computing: FCC technological advisory council (5G IoT working group)," 5 GPPP, Heidelberg, Germany, White Paper, 2018.

[9] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[10] A. Yousefpour et al., "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep. 2019. [Online]. Available: https://doi.org/10.1016/j.sysarc.2019.02.009

[11] *System Architecture for the 5G System V16.1.0*, 3GPP Standard TS 23.501, Aug. 2019.

[12] M. Patel, C. Chan, N. Sprecher, S. Abeta, and N. Neal, "Mobile-edge computing introductory technical white paper," Mobile-Edge Comput. Ind. Initiative, ETSI, Sophia Antipolis, France, White Paper, 2014.

[13] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017, doi: 10.1109/COMST.2017.2705720.

[14] J. Zhang, W. Xia, F. Yan, and L. Shen, "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing," *IEEE Access*, vol. 6, pp. 19324–19337, 2018, doi: 10.1109/ACCESS.2018.2819690.

[15] P. Ranaweera, A. D. Jurcut, and M. Liyanage, "Survey on multi-access edge computing security and privacy," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1078–1124, 2nd Quart., 2021, doi: 10.1109/COMST.2021.3062546.

[16] H. Wang, Z. Zheng, S. Xie, H. Dai, and X. Chen, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Services*, vol. 14, no. 4, pp. 352–375, 2018, doi: 10.1504/IJWGS.2018.10016848.

[17] S. Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[18] M. Rauchs et al., "Distributed ledger technology systems: A conceptual framework," *SSRN Electron. J.*, Jan. 2018, doi: 10.2139/ssrn.3230013. [Online]. Available: https://www.researchgate.net/publication/327568235_Distributed_Ledger_Technology_Systems_A_Conceptual_Framework

[19] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congress)*, 2017, pp. 557–564, doi: /10.1109/BigDataCongress.2017.85.

[20] T. Alladi, V. Chamola, R. M. Parizi, and K. -K. R. Choo, "Blockchain applications for industry 4.0 and industrial IoT: A review," *IEEE Access*, vol. 7, pp. 176935–176951, 2019, doi: 10.1109/ACCESS.2019.2956748.

[21] P. Tasca and C. Tessone, "A taxonomy of blockchain technologies: Principles of identification and classification," *Ledger J.*, vol. 4, pp. 1–39, Feb. 2019. [Online]. Available: https://ledger.pitt.edu/ojs/ledger/article/view/140

[22] M. S. Ferdous, M. J. M. Chowdhury, and M. A. Hoque, "A survey of consensus algorithms in public blockchain systems for crypto-currencies," *J. Netw. Comput. Appl.*, vol. 182, May 2021, Art. no. 103035. [Online]. Available: https://doi.org/10.1016/j.jnca.2021.103035

[23] T. Ncube, N. Dlodlo, and A. Terzoli, "Private blockchain networks: A solution for data privacy," in *Proc. 2nd Int. Multidiscip. Inf. Technol. Eng. Conf. (IMITEC)*, 2020, pp. 1–8, doi: 10.1109/IMITEC50163.2020.9334132.

[24] O. Dib, K. Brousmiche, A. Durand, E. Thea, and E. Hamida. "Consortium blockchains: Overview, applications and challenges." Sep. 2018. [Online]. Available: https://www.researchgate.net/publication/328887130_Consortium_Blockchains_Overview_Applications_and_Challenges

[25] D. Li, W. E. Wong, and J. Guo, "A survey on blockchain for enterprise using hyperledger fabric and composer," in *Proc. 6th Int. Conf. Dependable Syst. Appl. (DSA)*, 2020, pp. 71–80, doi: 10.1109/DSA.2019.00017.

[26] W. Azariah, F. A. Bimo, C.-W. Lin, R.-G. Cheng, R. Jana, and N. Nikaein, "A survey on open radio access networks: Challenges, research directions, and open source approaches," 2022, *arXiv:2208.09125*.

[27] "IEEE 5G and beyond technology roadmap; IEEE 5G initiative technology roadmap working group," 5 GPPP, Heidelberg, Germany, White Paper, 2018.

[28] L. Zhou, G. Wang, T. Cui, and X. Xing, "CSSP: The consortium blockchain model for improving the trustworthiness of network software services," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, 2017, pp. 101–107, doi: 10.1109/ISPA/IUCC.2017.00024.

[29] S. G. Sharma, L. Ahuja, and D. P. Goyal, "Building secure infrastructure for cloud computing using blockchain," in *Proc. 2nd Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, 2018, pp. 1985–1988, doi: 10.1109/ICCONS.2018.8663145.

[30] J. Ricci, I. Baggili, and F. Breitinger, "Blockchain-based distributed cloud storage digital forensics: Where's the beef?" *IEEE Security Privacy*, vol. 17, no. 1, pp. 34–42, Jan./Feb. 2019, doi: 10.1109/MSEC.2018.2875877.

[31] C. Xu, K. Wang, and M. Guo, "Intelligent resource management in blockchain-based cloud datacenters," *IEEE Cloud Comput.*, vol. 4, no. 6, pp. 50–59, Nov./Dec. 2017, doi: 10.1109/MCC.2018.1081060.

[32] K. Kotobi and M. Sartipi, "Efficient and secure communications in smart cities using edge, caching, and blockchain," in *Proc. IEEE Int. Smart Cities Conf. (ISC2)*, 2018, pp. 1–6, doi: 10.1109/ISC2.2018.8656946.

[33] C. Li and L.-J. Zhang, "A blockchain based new secure multi-layer network model for Internet of Things," in *Proc. IEEE Int. Congr. Internet Things (ICIOT)*, 2017, pp. 33–41, doi: 10.1109/IEEE.ICIOT.2017.34.

[34] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Security Privacy Workshops*, 2015, pp. 180–184, doi: 10.1109/SPW.2015.27.

[35] B. Mafakheri, T. Subramanya, L. Goratti, and R. Riggio, "Blockchain-based infrastructure sharing in 5G small cell networks," in *Proc. 14th Int. Conf. Netw. Service Manage. (CNSM)*, 2018, pp. 313–317.

[36] D. B. Rawat, M. S. Parwez, and A. Alshammari, "Edge computing enabled resilient wireless network virtualization for Internet of Things," in *Proc. IEEE 3rd Int. Conf. Collaboration Internet Comput. (CIC)*, 2017, pp. 155–162, doi: 10.1109/CIC.2017.00030.

[37] X. Ling, J. Wang, T. Bouchoucha, B. C. Levy, and Z. Ding, "Blockchain radio access network (B-RAN): Towards decentralized secure radio access paradigm," *IEEE Access*, vol. 7, pp. 9714–9723, 2019, doi: 10.1109/ACCESS.2018.2890557.

[38] K. Kotobi and S. G. Bilén, "Blockchain-enabled spectrum access in cognitive radio networks," in *Proc. Wireless Telecommun. Symp. (WTS)*, 2017, pp. 1–6, doi: 10.1109/WTS.2017.7943523.

[39] K. Kotobi and S. G. Bilen, "Secure blockchains for dynamic spectrum access: A decentralized database in moving cognitive radio networks enhances security and user access," *IEEE Veh. Technol. Mag.*, vol. 13, no. 1, pp. 32–39, Mar. 2018, doi: 10.1109/MVT.2017.2740458.

[40] M. Fukumitsu, S. Hasegawa, J.-Y. Iwazaki, M. Sakai, and D. Takahashi, "A proposal of a secure P2P-type storage scheme by using the secret sharing and the blockchain," in *Proc. IEEE 31st Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, 2017, pp. 803–810, doi: 10.1109/AINA.2017.11.

[41] K. Biswas and V. Muthukkumarasamy, "Securing smart cities using blockchain technology," in *Proc. IEEE 18th Int. Conf. High Perform. Comput. Commun. IEEE 14th Int. Conf. Smart City IEEE 2nd Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, 2016, pp. 1392–1393, doi: 10.1109/HPCC-SmartCity-DSS.2016.0198.

[42] J. Li et al., "Decentralized on-demand energy supply for blockchain in Internet of Things: A microgrids approach," *IEEE Trans. Comput. Social Syst.*, vol. 6, no. 6, pp. 1395–1406, Dec. 2019, doi: 10.1109/TCSS.2019.2917335.

[43] T. Fernández-Caramès, O. Blanco-Novoa, M. Suárez-Albela, and P. Fraga-Lamas, "A UAV and blockchain-based system for industry 4.0 inventory and traceability applications," *Proceedings*, vol. 4, no. 1, p. 26, Nov. 2018, doi: 10.3390/ecsa-5-05758.

[44] X. Zhang and X. Chen, "Data security sharing and storage based on a consortium blockchain in a vehicular ad-hoc network," *IEEE Access*, vol. 7, pp. 58241–58254, 2019, doi: 10.1109/ACCESS.2018.2890736.

[45] V. Ramani, T. Kumar, A. Bracken, M. Liyanage, and M. Ylianttila, "Secure and efficient data accessibility in blockchain based healthcare systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 206–212, doi: 10.1109/GLOCOM.2018.8647221.

[46] T. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–67, Apr. 2017.

[47] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[48] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: From concept to practice and beyond," *IEEE Commun. Mag.*, vol. 3, no. 53, pp. 80–88, Mar. 2015.
[49] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Trans. Mobile Comput.*, vol. 14, no. 12, pp. 2516–2529, Dec. 2015, doi: 10.1109/TMC.2015.2405539.
[50] L. Xiao, C. Xie, T. Chen, H. Dai, and H. V. Poor, "Mobile offloading game against smart attacks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2016, pp. 403–408, doi: 10.1109/INFCOMW.2016.7562110.
[51] N. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal auction for edge computing resource management in mobile blockchain networks: A deep learning approach," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.
[52] M. Liu, F. Yu, Y. Teng, V. Leung, and M. Song, "Distributed resource allocation in blockchain-based video streaming systems with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 695–708, Jan. 2019.
[53] B. Yang, X. Cao, J. Bassey, X. Li, T. Kroecker, and L. Qian, "Computation offloading in multi-access edge computing networks: A multi-task learning approach," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6, doi: 10.1109/ICC.2019.8761212.
[54] L. Xiao et al., "A reinforcement learning and blockchain-based trust mechanism for edge networks," *IEEE Trans. Commun.*, vol. 68, no. 9, pp. 5460–5470, Sep. 2020.
[55] M. Mukherjee, V. Kumar, A. Lat, M. Guo, R. Matam, and Y. Lv, "Distributed deep learning-based task offloading for UAV-enabled mobile edge computing," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2020, pp. 1208–1212, doi: 10.1109/INFOCOMWKSHPS50562.2020.9162899.
[56] J. Niu, S. Zhang, K. Chi, G. Shen, and W. Gao, "Deep learning for online computation offloading and resource allocation in NOMA," *Comput. Netw.*, vol. 216, Oct. 2022, Art. no. 109238. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.109238
[57] L. Chen and J. Xu, "Socially trusted collaborative edge computing in ultra dense networks," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–11.
[58] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 36, no. 3, pp. 574–585, Aug. 2018.
[59] X. He and S. Wang, "Peer offloading in mobile edge computing with worst-case response time guarantees," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2722–2735, Feb. 2021.
[60] X. He, S. Wang, and X. Wang, "Providing worst-case latency guarantees with collaborative edge servers," *IEEE Trans. Mobile Comput.*, early access, Dec. 7, 2021, doi: 10.1109/TMC.2021.3133306.
[61] A. Vera-Rivera, A. Refaey, and E. Hossain, "A blockchain framework for secure task sharing in multi-access edge computing," *IEEE Netw.*, vol. 35, no. 3, pp. 176–183, May/Jun. 2021, doi: 10.1109/MNET.011.2000497.
[62] A. Vera-Rivera, A. Refaey, and E. Hossain, "Blockchain-based collaborative task offloading in MEC: A hyperledger fabric framework," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2021, pp. 1–6.
[63] A. Vera-Rivera, A. Refaey, and E. Hossain, "Task sharing and scheduling for edge computing servers using hyperledger fabric blockchain," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, 2021, pp. 1–6.
[64] M. Noback, "The acyclic dependencies principle," in *Principles of Package Design: Creating Reusable Software Components*. Berkeley, CA, USA: Apress, 2018, pp. 185–216.
[65] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *Proc. IEEE 26th Int. Symp. Model. Anal. Simulat. Comput. Telecommun. Syst. (MASCOTS)*, 2018, pp. 264–276.
[66] E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Recent developments in deterministic sequencing and scheduling: A survey," in *Deterministic and Stochastic Scheduling*. Dordrecht, The Netherlands: Springer, 1982, pp. 35–73, doi: 10.1007/978-94-009-7801-0_3.
[67] L. Perron and V. Furnon. *OR-Tools, V9.4*. Aug. 2022. Google. [Online]. Available: https://developers.google.com/optimization/
[68] M. Masdari, S. Jabbehdari, M. R. Ahmadi, S. M. Hashemi, J. Bagherzadeh, and A. Khadem-Zadeh, "A survey and taxonomy of distributed certificate authorities in mobile ad hoc networks," *J. Wireless Commun. Netw.*, vol. 112, p. 112, Sep. 2011. [Online]. Available: https://doi.org/10.1186/1687-1499-2011-112
[69] A. Vera-Rivera. "Task sharing service demo repository." 2022. [Online]. Available: https://github.com/angelovera/TaskSharingServiceDemo

ANGELO VERA-RIVERA (Student Member, IEEE) was born in Guayaquil, Ecuador. He received the B.Sc. degree in electronics and telecommunications from the Escuela Superior Politecnica del Litoral, Ecuador, in 2011, the first M.Sc. degree in telecommunications from George Mason University, USA, in 2015, and the second M.Sc. degree in electrical and computer engineering from the University of Manitoba, Canada, 2022, under the supervision of Dr. Ekram Hossain working with the Wireless Communications, Networks, and Services Laboratory. His research interests are centered around the application of blockchain technologies into the architecture of new generation communication networks.

EKRAM HOSSAIN (Fellow, IEEE) is a Professor and the Associate Head (Graduate Studies) with the Department of Electrical and Computer Engineering, University of Manitoba, Canada. His current research interests include design, analysis, and optimization of wireless networks with emphasis on beyond 6G cellular networks. He received the 2017 IEEE ComSoc Technical Committee on Green Communications and Computing Distinguished Technical Achievement Recognition Award "for outstanding technical leadership and achievement in green wireless communications and networking." He served as the Director of Magazines for IEEE ComSoc from 2020 to 2021. He served as the Editor-in-Chief for the IEEE Press from 2018 to 2021 and the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS from 2012 to 2016. Since 2022, he has been serving as the Director of Online Content for IEEE ComSoc. He was listed as a Clarivate Analytics Highly Cited Researcher in Computer Science from 2017 to 2022. He was an Elected Member of the Board of Governors of the IEEE ComSoc from 2018 to 2020. He was elevated as a Fellow of IEEE "for contributions to spectrum management and resource allocation in cognitive and cellular radio networks." He is a member (Class of 2016) of the College of the Royal Society of Canada and a Fellow of the Canadian Academy of Engineering and the Engineering Institute of Canada.

AHMED REFAEY HUSSEIN (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from Alexandria University, Alexandria, Egypt, in 2003 and 2005, respectively, and the Ph.D. degree from Laval University, Quebec, QC, Canada, in 2011. He is currently an Assistant Professor with the University of Guelph, Guelph, ON, Canada, and an Adjunct Research Professor with Western University, London, ON, Canada. Previously, his positions included: a Professional Researcher with the LRTS Laboratory, Laval University, in the field of wireless communications hardware implementations from 2007 to 2011; a Postdoctoral Fellow with the Department of Electrical and Computer Engineering, Western University, from 2012 to 2013; a Senior Embedded Systems Architect with the Research and Development Group, Mircom Technologies Ltd., Toronto, ON, Canada, from 2013 to 2016; and an Associate Professor with Manhattan College, New York, NY, USA, from 2016 to 2021. Prior to joining Laval University, he was a System/Core Network Engineer leading a team of junior engineers and technicians in the telecom field in the three prominent companies of Fujitsu, Vodafone, and Alcatel-Lucent, all in Cairo, Egypt. He is the author or coauthor of more than 70 technical articles, one patent granted, and three patent applications addressing his research activities. He served as the Central Area Chair for IEEE R7 in 2021.