Decentralized Allocation of Geo-distributed Edge Resources using Smart Contracts

Jinlai Xu[†], Balaji Palanisamy[†], Qingyang Wang[‡] Heiko Ludwig[§] and Sandeep Gopisetty[§]
[†]School of Computing and Information, University of Pittsburgh, Pittsburgh, PA 15213, USA
Email: jinlai.xu@pitt.edu, bpalan@pitt.edu
[‡]Computer Science and Engineering, Louisiana State University, Baton Rouge, LA 70803, USA
Email: qywang@csc.lsu.edu
[§]Almaden Research Center, IBM Research, San Jose, CA 95120, USA
Email: hludwig@us.ibm.com, Sandeep.Gopisetty@us.ibm.com

2022 22nd International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-6654-9956-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CCGRID54584.2022.00052

***Abstract*—In the Internet of Things (IoT) era, edge computing is a promising paradigm to improve the quality of service for latency sensitive applications by filling gaps between the IoT devices and the cloud infrastructure. Highly geo-distributed edge computing resources that are managed by independent and competing service providers pose new challenges in terms of resource allocation and effective resource sharing to achieve a globally efficient resource allocation. In this paper, we propose a novel blockchain-based model for allocating computing resources in an edge computing platform that allows service providers to establish resource sharing contracts with edge infrastructure providers *apriori* using smart contracts in Ethereum. The smart contract in the proposed model acts as the auctioneer and replaces the trusted third-party to handle the auction. The blockchain-based auctioning protocol increases the transparency of the auction-based resource allocation for the participating edge service and infrastructure providers. The design of sealed bids and bid revealing methods in the proposed protocol make it possible for the participating bidders to place their bids without revealing their true valuation of the goods. The truthful auction design and the utility-aware bidding strategies incorporated in the proposed model enables the edge service providers and edge infrastructure providers to maximize their utilities. We implement a prototype of the model on a real blockchain test bed and our extensive experiments demonstrate the effectiveness, scalability and performance efficiency of the proposed approach.**

***Index Terms*—Edge Computing, Blockchain, Smart Contract, Incentive Design, Resource Allocation**

I. Introduction

In the Internet of Things (IoT) era, edge computing is a promising paradigm to improve the quality of service for latency sensitive applications by filling gaps between the IoT devices and the cloud infrastructure. Edge/Fog Computing [1]–[8] provides an additional layer of computing infrastructure for storing and processing data at the edge, allowing low latency applications to meet their response time requirements effectively. Edge service providers (SPs) use edge computing resources available near the end devices (e.g., smart Internet of thing devices) for scheduling computation tasks to meet the strict latency requirement of edge computing applications. However, the state-of-the-art edge platforms are specifically designed for customized applications for a specific service provider (SP), rather than a public service/platform that can be used for various applications by users. While such a model reduces the complexity of managing the edge computing resources, it leads to significant inefficiencies including lower cost-effectiveness. In the current model, the resources available at the edge are not completely utilized because of the isolation that exists between service providers. Besides the security considerations, the technical barriers (e.g., elasticity of the distributed edge computing platform, seamless computation migration between the cloud and the edge and geo-distributed resource management) prevent public edge resource sharing. This becomes further challenged with financial considerations such as profitability to run and participate in a public edge federation, guaranteeing fairness of the platform, monitoring and auditing the resource usage and payments.

Blockchain-based smart contracts [9] provide a promising approach to build decentralized, transparent, trustworthy, and self-organized ecosystems between multiple independent parties. The smart contracts can act as a trusted third-party to execute the resource allocation algorithms between the edge service providers and the edge infrastructure providers. A blockchain-based resource allocation scheme can enhance the data and execution transparency of the resource allocation process and enable a transparent payment distribution scheme among the providers in an autonomous manner. Besides increasing the transparency, smart contracts can make it easier to establish the relationship and trust between the providers. Designing a blockchain-based transparent resource allocation model incurs several challenges including designing appropriate incentive structures to make sure that all the participants can benefit by participating in the ecosystem.

In this work, we propose, design and implement a decentralized platform for sharing geo-distributed edge resources among multiple entities. This paper makes the following novel contributions:

- We first propose a decoupled decentralized resource allocation model that manages the allocation of computing resources distributed at the edges to the service providers that have application demand to use those resources.
- We propose a sealed bid double auction protocol based on decentralized smart contracts with a two-phase sealed bidding and revealing mechanism.

978-1-6654-9956-9/22/$31.00 ©2022 IEEE
DOI 10.1109/CCGrid54584.2022.00052

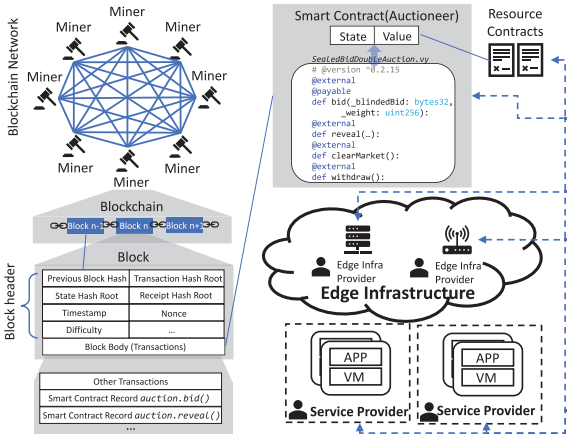

Fig. 1. Blockchain-based Edge Resource Sharing

- Based on the protocol, we develop a decentralized auction-based resource sharing contract establishment and allocation mechanism that ensures truthfulness and utility-maximization for the providers.
- Finally, we implement a prototype of the proposed model on a real blockchain test-bed and our extensive experiments demonstrate the effectiveness, scalability and performance efficiency of the proposed approach.

II. Background and Motivation

In the IoT era, the demands for low-latency computing for latency-sensitive applications (e.g., location-based augmented reality games, real-time smart grid management, real-time navigation) have been growing rapidly. Edge Computing provides an additional layer of infrastructure to fill latency gaps between the IoT devices and the back-end cloud computing infrastructure.

In current edge computing models, we have cloud providers that provide geo-distributed cloud infrastructure or edge computing infrastructure as a service. With the development of cloud-native techniques [10], it makes the migration of services (micro-services, containers) easier between datacenters including the micro datacenters (MDCs) deployed at the edge layer. Current models also include cloud services that are extended to support edge computing. For instance, Amazon Greengrass [11] supports Amazon Lambda [12] and other AWS services on the edge devices and Azure IoT Edge [13] extends the Azure cloud services to the edge devices. The extension motivates the service providers to invest and deploy their own edge infrastructure at the network edge. However, on one hand, current models operate by restricting each service provider to only utilize their own edge infrastructure resources for providing service. It creates a strong barrier between the providers and makes the infrastructure investment not only inefficient but also redundant. On the other hand, the previous fixed-price model of cloud computing may not be suitable to be used in the highly diverse and dynamic edge computing environment.

Sharing edge infrastructure has significant benefits to optimizing resource usage cost and meeting strict latency requirements. Geo-distributed edge infrastructures that cooperate together to optimize resource allocation can build a seamless geo-distributed edge federation platform to provide infrastructure to a wide range of edge computing applications (e.g. virtual reality, smart city, big data analytic) that have strict latency and bandwidth requirements.

In this work, we design a decentralized mechanism that enables resource sharing among the service providers and the edge infrastructure providers (Figure 1). The proposed approach implements a long-term persistent resource sharing scheme at the edge using decentralized blockchain networks. In the remaining part of this section, we describe the background of edge resource sharing and blockchain-based smart contracts.

A. Edge Resource Sharing

We assume that the resources are organized in a layered architecture [14] where the edge infrastructure acts as the middle layer between the cloud infrastructure and the smart things. The dense geo-distributed edge infrastructure includes the MDCs and the smart gateways placed at the edge of the network, which is located one hop away from the end devices. We model the edge resource sharing platform similar to the model described in [8]. The platform includes (i) potential buyers namely service providers (SPs) who are responsible to provide services directly to the end users and want to lease edge resources on demand to increase their revenues, and (ii) potential sellers namely edge infrastructure providers (EIP) who operate either the MDCs or the smart gateways that support multi-tenant resource allocation by running containers.

To model the resource sharing problem at the edge, we assume that there are $|N|$ SPs that require edge computing infrastructure to support their services. For simplicity, we assume that for each SP $i \in N$, where N is the set of the SPs, there is a quantifiable service demand of the SPs in each geographic region for every discrete time slot of a day. The resource requirement is represented by the application container [15], [16] (a configured virtual machine integrated with the service software), which has several requirements such as CPU consumption, memory size, network bandwidth and latency requirement. We use $\lambda_i^p(\tau)$ to represent the workload coming from a particular location p in time slot τ for SP i as shown in Figure 2. We assume that there are several EIPs and each of them handles a large number of highly geo-distributed MDCs and smart gateways that represent the edge computing resources. We assume that each MDC or smart gateway can act as a virtual node $d \in D$, which can support the deployment of several containers. For simplicity, we assume that every container consumes an equal amount of resources for running the application service. The capacity of the overall node is denoted as C_d and it represents the number of containers that can be run on the node. The final resource allocation decision can be simplified as a mapping between the edge resources $C_d, \forall d \in D$, and the workloads, $\lambda_i^p(\tau), \forall i \in N$, which decides the actual node that run the containers to serve the corresponding workloads from a location p. The decision

problem is NP-hard [8] and in this work, we simplify the problem by first delineating non-overlapping regions and then dividing the continuous sharing time into non-overlapping time slots (e.g., one hour). Thus, each trade (auction) is handled for a particular region $r \in R$ and for a particular time slot τ, which allows the problem to be solved using the proposed auction mechanism (Section III-C).

The key challenges of the edge resource sharing problem are two folds: (i) the geo-distributed nature and the distributed ownership of the edge infrastructure make it hard to centralize the resource allocation decision and (ii) the competitive relationship between the EIPs and the SPs make it challenging to guarantee efficiency and fairness. To tackle these challenges, we employ blockchain-based smart contracts to deploy a truthful auction that automatically processes the bids from the potential buyers and sellers to generate resource contracts between them guaranteeing both efficiency, fairness, and decentralization at the same time.

B. Blockchains and Smart Contracts

Blockchain is a distributed ledger that stores transaction records as a chain of blocks maintained by a set of miners in the decentralized blockchain network (Figure 1). The miners mine the blocks to include the transactions to form the blockchain for the state in which all miners reach an agreement through a consensus protocol (e.g., proof-of-work (PoW), or proof-of-stake (PoS)). The architecture of blockchain makes it possible to achieve decentralization, integrity, auditablility, transparency and high availability at the same time. Smart contracts are built on top of blockchain and they allow user-defined programs to be executed on the blockchain. More specifically, the smart contract can be treated as a program deployed on the blockchain network, which resides at a specific address (generated when deploying) on the blockchain, including algorithms (functions within a contract) and data (the state of the smart contract) as shown in Figure 1. To interact with smart contracts, there are two ways: (i) retrieving the state or data from the smart contracts which can be directly restored from the blockchain data without sending transactions, and (ii) change the state of the smart contracts which require calling the functions of the smart contract by sending transactions and the execution is completed after the transaction is included in the blockchain network. Ethereum is well-known blockchain network that supports smart contracts. Ether is the cryptocurrency used on Ethereum. It is held in and can be transferred between accounts including externally owned accounts (EOAs) and contract accounts (CAs). An EOA is determined by a unique public-private key pair owned by an individual who can use the private key to sign the transactions sent from the account. CAs are different than EOAs. Each CA does not have a key pair and is associated with a deployed smart contract that is activated (deployed) by an EOA. To execute the smart contract, the EOA that deploys a new smart contract or calls a function of a deployed smart contract needs to pay Gas [9] included in the transaction. Gas can be exchanged from Ether. A transaction in Ethereum is

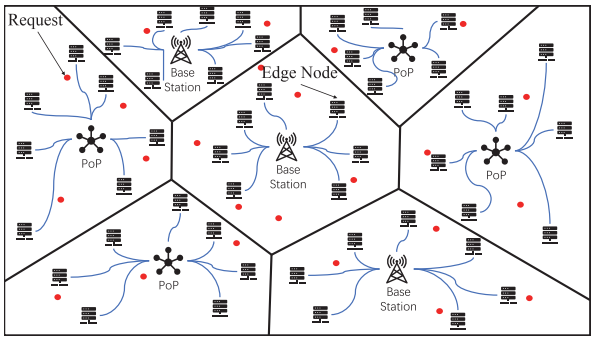

Fig. 2. Regions

a build-in instruction signed by an EOA. Each transaction specifies several information including the sender's address, the receiver's address and *data* (e.g., smart contract bytecode, and a function call with the arguments). Each function call and its input are included in the transaction so that the output can be verified by multiple miners with the same input and the given program. The correctness can be guaranteed by the miners and the consensus protocol of the blockchain.

In this work, the proposed resource sharing (auction) protocol is enforced without trusted third-parties using smart contracts. Additionally, the proposed truthful auction is designed to guarantee that the bidders will bid with their true valuation on the goods and thus, it will reduce the complexity of the auction algorithm. The resource contracts that are persistent and distributed on the blockchain can be used for assessment, billing, and auditing. To cooperate with the geo-distributed utility-aware task scheduling, the edge resource sharing platform can be seamlessly integrated with the existing utility-aware cloud platform to handle a wide range of applications with different requirements.

III. Smart Contract-based Edge Resource Allocation

We design the proposed edge resource sharing platform using blockchain-based smart contracts. The blockchain acts as the decentralized ledger that stores the trade information (e.g., when and where the resource is shared), and the procedure (smart contracts) of the trade between the buyers and sellers transparently for all the participants.

A. System Architecture

The architecture of the proposed edge resource sharing platform is shown in Figure 1. The basic functionality of the platform is to match the supply and demand of the edge resources based on the auction algorithm deployed in the smart contracts. To make the process decentralized, we employ blockchain-based smart contracts to act as the auctioneer.

As shown in Figure 1, we can see that the decentralized consensus and mining of the new blocks are controlled by the miners connected to the blockchain network. Any node can download the client of the blockchain network and be a miner of the network. The public blockchain can be audited by anyone who participates in the blockchain network and accepts the broadcast. The blockchain is established by a sequence of

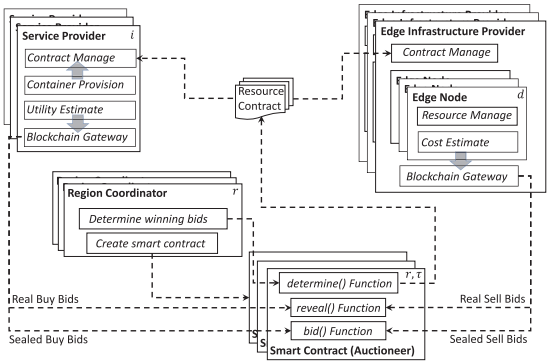

Fig. 3. Edge Resource Sharing Framework

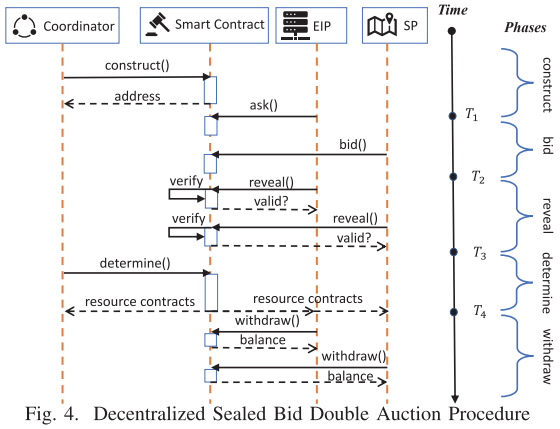

Fig. 4. Decentralized Sealed Bid Double Auction Procedure

blocks, each of them is generated by the consensus mechanism and connected to its parent block by its hash value. The block stores all the information related to the state changes of the blockchain (e.g., transactions between accounts, and modifications of values in the smart contracts) in their block body as records of transactions. The smart contracts deployed on the blockchain can achieve automatic execution of the algorithms and guarantee the correctness of transactions. In the proposed system, we assume that there are four types of entities namely the service providers (SPs), Edge Infrastructure Providers (EIPs) discussed in Section II, Region Coordinators, and smart contracts as shown in Figure 3.

We use the notion of divided regions to reduce the complexity of matching the latency or location requirement of the edge application to the resources available in a certain area. We assume that the map corresponding to the geographic area is divided into $|R|$ sub-regions, where R is the set of all the regions. The region division can be generated by negotiation between the EIPs and SPs, which can either be based on the location of base stations, Point of Presences (PoPs) of Internet Service Providers (ISPs) or based on administrative divisions (e.g., counties) as shown in Figure 2. Each region $r \in R$ only includes a specific area and there is no overlap between the regions. We also assume that the expected workload distribution for each time slot τ is $\lambda_i^r(\tau)$ for SP i in the region r. The workload distribution contains all the workloads coming from the region. We use $\lambda_i^p(\tau)$ to represent the workload coming from a particular position p in region r. As we primarily consider the workload which needs real-time service, $\lambda_i^p(\tau)$ is often the upper bound of the workload during the time slot τ from position p. In each region, there can be multiple nodes that serve as the infrastructure. We use E_r to represent the list of nodes which serves in region r. In addition, the nodes that serve in one region can guarantee the lowest possible latency of placing the services of the SPs with an average latency, l_r, as they either directly connect to the PoP or base station of the region, which is one hop from the end devices that send the requests. With the above assumptions, the edge nodes can be either MDCs, smart gateways, or even mega datacenters that satisfy the placement requirement for a certain region.

We use smart contracts to run the resource trading between the SPs and the EIPs in each region. The smart contract acts as the trusted third party that uses the predefined auction algorithm to decide the winning bids and establish the resource contracts. However, as we need both the resource buyers (SPs) and the sellers (EIPs) to bid in the (double) auction, both of them are not suitable to create or handle the smart contract as the owner. Therefore, we assume that there is a region coordinator for each region. The coordinator creates the smart contract for the auctions based on the rules (e.g., when the participants can bid, and what is the time duration for the resource contracts for a particular auction). It notifies the participants the information of the smart contract (e.g., address, and interfaces), monitors the smart contract and calls the function of the smart contract to determine the winning bids. The coordinator is the owner of the smart contract and the cost of running the smart contract is paid either by registering the auction or from the subsidy of the auction.

The auction algorithm and the status is published in the smart contract. The participants and the coordinator can audit the status of the smart contract from the blockchain data. The smart contract acts as the auctioneer and runs the auction automatically from the predefined auction algorithm and decides the winning bids. Then, based on the policy, the resource contracts are established and recorded in the blockchain for further accounting and auditing when the resources are used.

B. Decentralized Sealed Bid Double Auction Protocol

In this section, we present the proposed resource allocation techniques for EIPs and SPs to establish relationships (resource contracts) with each other and explain how smart contracts help in this process.

We design a sealed bid double auction using smart contracts on the blockchain that enables the EIPs and the SPs to bid in the auction with their sealed bids. We assume that for each region r and each time slot τ, there is an auction that decides which EIP and SP pairs trade with each other. We note that all participants and the coordinator can interact with the blockchain network using blockchain gateway services such as Infura (https://infura.io/). We assume that each EIP and SP have at least a client that is responsible to interact with the region coordinators, resource management, and the smart contracts to control the process of evaluating the resource

```
# @version ^0.2.15
# SealedBidDoubleAuction.vy
struct Bid:
    bidder: address
    blindedBid: bytes32
    deposit: uint256
    weight: uint256
    value: uint256
...
# Auction parameters
coordinator: public(address)
biddingEnd: public(uint256)
revealEnd: public(uint256)
...
# State of the bids
asks: public(HashMap[address, Bid[MAX_BIDS]])
bids: public(HashMap[address, Bid[MAX_BIDS]])
askCounts: public(HashMap[address, uint256])
bidCounts: public(HashMap[address, uint256])
validAsks: public(Bid[MAX_CONTRACTS])
validAskCount: public(uint256)
validBids: public(Bid[MAX_CONTRACTS])
validBidCount: public(uint256)
# Allowed refund map (withdraw or payment)
pendingReturns: public(HashMap[address, uint256])
```

Fig. 5. Smart Contract Initial Parameters

valuation. They interact with the smart contract (auctioneer) and record and report the established resource contracts to the internal resource management for resource allocation and task scheduling.

The auction procedure consists of five phases as shown in Figure 4:

Phase 1. *Construct*: includes the registration and smart contract construction. Before the auction smart contract is created, all the EIPs and SPs that want to participate need to register their accounts to the region coordinator through the steps shown below. The region coordinator will also record the account information.

Smart Contract Construction

Input: region id r, resource sharing time slot τ, region coordinator account r^{addr}
Output: smart contract address z
Before time T_1, for each region r and each time slot τ:

1. The region coordinator creates the smart contract by calling $z = r^{addr}.deploy(Z, T_1, T_2, T_3, r, \tau)$, where Z is the class definition of the resource sharing auction.
2. The region coordinator waits for the transaction to be included in the blockchain network and gathers the address z for the smart contract.
3. The region coordinator broadcasts the smart contract tuple $< z, r, \tau >$ to all the participants registered.

We use the notations T_1, T_2, T_3, T_4 to represent the end time of the first four phases (*construct*, *bid*, *reveal*, and *determine*) as shown in Figure 4. When handling the construction of the smart contract, the region coordinator will obey the rules of phase periods. For example, for the *construct* phase, the region coordinator will call the constructor function of the smart contract before T_1. It is worth noting that T_1, T_4 are enforced by the auction protocol which needs to obey the region coordinator but T_2, T_3 can be enforced directly by the smart contract which is included in the program of the smart contract. Based on the negotiated time slot length of the edge resource sharing period (for example, in an one hour period), the region coordinator creates a smart contract for each resource sharing period. Several parameters are initialised together with the smart contract as shown in Figure 5. It written in Vyper [17] to include the details. We define a `Bid` structure at the beginning of the smart contract and we initialize the map between the sellers and the sell bids (`asks`), the map between the buyers and the buy bids (`bids`) and the counts of them. After the smart contract for the auction is created, the region coordinator will get an address for the smart contract. Then, both the address and other related information (e.g., region id, and time slot) are broadcasted to all the participants.

Phase 2. *Bid*: After the auction smart contract is broadcasted to all the participants, the participant can fetch the auction calendar (e.g., when the participants can bid, and reveal) from the smart contract. It is as mentioned above as noted by T_1, T_2, T_3. When the bidding period is started, all the participants who registered can bid to the smart contract. For each bid, the EIP or SP sends the bid by interacting with the smart contract by sending a transaction including the function for bidding, the blinded bid (hash value of the entire bid with a randomly generated secret), the number of containers requested or available, and the deposit to the smart contract. We omit the procedure for the seller (EIPs) to place their sell bids (asks) as it is similar to the bid procedure shown below. The deposit of the sell bid (ask) will be used to guarantee that the resources are preserved for the resource contracts during the effective time slot.

Bid Procedure

Input: smart contract address z, bid value b, weight c, SP account i^{addr}
After time T_1, before time T_2:

1. SP i generates the blinded bid $\beta = hash(b, c, \alpha)$ with a randomly generated secret α.
2. SP i decides a deposit value, which is $\gamma = b * c + random(b * c)$.
3. SP i uses its registered account i^{addr} to send the bids by interacting with the smart contract z by calling $z.bid(\beta, c, \{"from" : i^{addr}, "value" : \gamma\})$, where the entries in the brackets describe the corresponding entries included in the transaction.
4. The smart contract z records the blinded bid information as a tuple, $< \beta, c, i^{addr}, \gamma >$.
5. SP i waits for the transactions to be verified and included in the blockchain network. Then, it record the bid in the local bid array $\mathbf{B}_i$ including tuples of the true bids, $< b, c, \alpha >$.

Phase 3. *Reveal*: in the *reveal* phase, the participant needs to interact with the smart contract to reveal the bids they sent by sending the real bid to be verified by the smart contract. The bid value, weight and secret will be sent to the smart contract for verification. If the stored hash value matches the hash value of the above tuple, the bid is valid and will be reserved for the auction. Otherwise the bid will be removed and the corresponding deposit will be appended to the withdraw fund

list. We omit the procedure for the seller (EIPs) to reveal their sell bids (asks) as it is similar to the reveal procedure of the buyers (SPs) shown below. It is worth noting that, in the *reveal* phase, it is possible for the bidders to cancel the previous bids by sending a wrong bid value to the smart contract in the corresponding bid entry. Therefore, we do not implement the cancellation functionality in the smart contract and we leave it to the client to implement it. As only the bidder has the private key to sign its own bids, the authentication of the blockchain network can make sure that the denial of service attack (e.g., send the wrong bids to the smart contract to cancel others bids) is hard to conduct.

Reveal Procedure

Input: smart contract address z, bid array $\mathbf{B}_i$, SP account i^{addr},
After time T_2, before time T_3:

1. SP i sends the bid array $\mathbf{B}_i$ where each entry includes the bid value b, weight c, secret α of the bid, to the smart contract for verification by calling $z.reveal(\mathbf{B}_i, \{"from" : i^{addr}\})$.
2. Smart contract z verifies each bid by the blinded bid hash β placed in the *bid* phase. If the buy bid matches the hash value and the deposit is larger than the total bid value ($\gamma \geq b * c$), it will be appended to the valid buy bid array, B, stored in the smart contract shown as an array `validBids` in Figure 5.

Phase 4. *Determine*: In the winner determination phase, the auctioneer (smart contract) decides the winning bids. For simplicity, we omit the region id r and time slot τ in the following discussion. We use a binary notation x_b to denote whether the buy bid b wins or not ($x_b = 1$ wins, and vice versa). Similarly, x_s denotes whether the sell bid (ask), s, wins or not. We denote the buy price as π_b and sell price as π_s. Similarly, we denote the auction result as two sets, $X_s = \{x_s | s \in S\}$ and $X_b = \{x_b | b \in B\}$. Each entry of the set determines the decision of one sell or buy bid in the auction. Besides the winners, the algorithm also sets the map (shown as `pendingReturns` in Figure 5) between the bidders and the pending withdraw amounts, which determines how much fund can be withdrawn for each bidder including the bid deposit of the fail bids and the overvalued deposit of the winning bids. The resource contracts are also established in this phase based on the results of the auction. Each of them can be denoted as a tuple $< i, \mathbf{D}, \mathbf{C}, \tau, \pi_b, \pi_s >$, where $\mathbf{D}$ is the list of sellers (edge nodes) that provides the resources to SP i in the resource contract and $\mathbf{C}$ is the array of the number of containers provided by each seller.

Winner Determination Procedure

Input: smart contract address z
After time T_3, before time T_4:

1. The region coordinator calls $z.determine()$, the predefined auction algorithm, in the smart contract to decide the winning bids, which is discussed in Section III-C. The decision X_b and X_s are stored in the smart contract along with the initial resource contracts.

Phase 5. *Withdraw*: After the auction is closed and the resource contracts are established, the participants can withdraw their remaining funds (e.g., the excess value deposit and the deposit of the fail bids) from the smart contract.

Withdraw Procedure

Input: smart contract address z, bidder's account a
For each bidder (either EIP or SP):

1. The bidder calls $z.withdraw()$ with its account a.
2. Smart Contract z verifies the available withdraw amount `pendingReturns[a]` defined in Figure 5. If `pendingReturns[a]>0`, the fund will be sent back to account a and set `pendingReturns[a]=0` to avoid double withdrawal.

C. Auction Algorithm

From the Myerson–Satterthwaite theorem [18], we can see that there are no auction algorithms that can satisfy all of the four auction properties at the same time namely: (i) Individual Rationality, which means that no participants should lose from bidding in the auction, (ii) Weak Balanced Budget, which means that the auctioneer will not subsidize the auction, (iii) Truthfulness that ensures that bidding with true valuation is the dominant strategy of all the bidders and (iv) Economic efficiency ensures that the good should be finally allocated to the bidder who values it the most. However, there are auction algorithms that can satisfy three of the properties with a bounded loss on the remaining one. McAfee mechanism [19] can satisfy individual rationality, weak balanced budget, truthfulness with a bounded loss of economic efficiency ($1/\min(|B|, |S|)$ in our problem). In our work, we use the McAfee mechanism in the auction design. The truthfulness property guarantees that for the participants whose objectives are to maximize their utilities, the dominant bidding strategy is to bid by their true valuation. Based on the above assumption, we first define the utility of the SP and EIP.

Utility of Service Provider: We model the utility of the SP to run the service at the edge. Here, we consider services having higher requirements for latency such as location-based augmented reality games [20] and intelligent traffic light control [21]. The utility gain of the SP can be expressed by the gain in changing the execution of the real-time service from the cloud to the edge which can be represented by the function:

$$v_i^p(\tau) = f(l_r) - f(l_{pi}(\tau)) \quad (1)$$

where $f(l)$ is a function which estimates the utility that can be obtained by providing the service with a latency l. We assume that the function is a non-increasing function related to the latency which means that when the latency is increased, the utility will decrease. Here $l_{pi}(\tau)$ represents the latency between the mega datacenter of SP i and the position p. We note that the utility gain can be also modeled by other criteria such as the bandwidth cost (e.g., when moving an aggregator operator to the edge to reduce the overall bandwidth cost of moving the data to the cloud).

Utility of Edge Infrastructure Provider For the EIP, its objective is to earn higher revenue by providing the infrastructure to SPs. Therefore, the utility for the EIP is obviously the profit that it can obtain by renting the resource to the SPs. The true

valuation of the resource for the EIP can be defined using the operating cost of the resources. For each node, the unit operating cost function $Cost_d(\tau)$ can be defined as the ratio of the sum of the operating cost of each server and the capacity of the node:

$$Cost_d(\tau) = \frac{\sum_m^{M_d} Cost_d^m(\tau)}{C_d} \quad (2)$$

where $Cost_d^m(\tau)$ is the fluctuating operating cost of server m in node d in time slot τ. To determine the winning bids, we extend the McAfee mechanism [19] by allowing each bid to contain both the unit price and the number of containers at the same time. The group bidding method can save a significant amount of cost when the auction is running on the smart contract.

The auction algorithm is shown in Algorithm 1. As we can see, the time complexity is $O(max(|B|log|B|, |S|log|S|))$. We omit the region id r and the time slot τ in the algorithm definition. In the algorithm, we can see that the buy bids and sell bids (asks) are sorted in their natural ordering (ascending order for sell bids and descending order for buy bids). Then, we find the break-even index by accumulating either from the buy bids or sell bids (asks) by counting the number of containers. When the break-even index is found (the next buy bid price is lower than the next sell bid price), the supply is filled (all the possible containers are sold) or we meet the end of the bid array. Then based on the McAfee mechanism, we decide the winning bids and the final selling and buying prices.

Algorithm 1 Algorithm for determining the winning bids

procedure DETERMINE$(B, S) \rightarrow \pi_s, \pi_b, X_s, X_b$
  sort B in descending order by the bid price
  re-index B as $B = \{b_i, c_i | i \in [1, |B|]\}$
  sort S in ascending order by the bid price
  re-index S as $S = \{s_j, c_j | j \in [1, |S|]\}$
  set the overall supply $C_r = \sum_{d \in E_r} C_d$
  set current buy price b as the first bid (highest price) in B
  set the sell price s as the first sell bid (lowest ask) in S
  set number of buying containers $h = 0$, and selling containers $k = 0$
  set current index $i = 1, j = 1$
  while $True$ **do**
    if $h \geq C_r$ **then**
      $h = C_r$
      break
    else if $i + 1 > |B|$ or $j + 1 > |S|$ or $b_{i+1} < s_{j+1}$ **then**
      break
    if $h > k$ **then**
      $s = s_j$, $k+ = c_j$, $x_j = 1$
      $j++$
    else
      $b = b_i$, $h+ = c_i$, $x_i = 1$
      $i++$
  $\rho = (b_{i+1} + s_{j+1})/2$
  if $b \geq \rho \geq s$ **then**
    $\pi_s = \pi_b = \rho$
  else
    $x_i = 0, x_j = 0$
    $\pi_b = b_i$
    $\pi_s = s_j$

D. Smart Contract Implementation

We implement the smart contract by Vyper [17], which can run on any blockchains that support Ethereum Virtual Machine (EVM) (such as Ethereum, Hyperledger Fabric, etc.). Our choice of using Vyper is due to its security, auditability and being predictable by implicitly limiting the features of the language such as recursive function calls, which may lead to unpredictable results when interacting with the smart contract. As Vyper does not support the dynamic array, we set the size of all the arrays that appear in the smart contract as 64. Limiting the number of bids can decrease the cost of establishing the smart contract and running the functions.

TABLE I
AUCTION SCHEDULE EXAMPLE FOR THE RESOURCE USAGE IN 15:00 - 16:00 9/30

Phase	Time	Participants/Executor
Construct	0:00 9/29	coordinator
Bid	0:00-16:00 9/29	EIPs and SPs
Reveal	16:00-23:00 9/29	EIPs and SPs
Determine	23:00-23:59 9/29	coordinator
Withdraw	after 23:59 9/29	Everyone

As our resource contracts can be established before the Service Providers actually use the resources, it provides adequate time for the coordinator to conduct the auction. As shown in Table I, the auction can be handled a day ahead and the two phases can be scheduled with in certain time ranges. As the time range is sufficient for each participant to interact with the smart contract and to wait for the transaction to be included in a block, and verify the transaction, the gas price can be set with a low priority fee or even without setting the priority fee to save the overall cost.

E. Resource Contract

After the auction is cleared, the resource contracts for time slot τ are established. The length of the time slot τ can be negotiated by the EIPs and SPs to determine an appropriate granularity for the resource allocation by considering both low-cost (e.g., the cost of running the auction on the smart contract) and efficiency for placing services (e.g., minimizing the migration when the resource contracts are expired). We assume the length of the time slot is one hour, and the auction will be handled on the previous day when the resource will be used. We present an example in Table I. After the winning bids are determined, the resource contracts are built between the EIP and SP pairs one by one, and the buyer i, the sellers $\mathbf{D}$ (the nodes provide the resources), the buying price π_b, selling price π_s, the array of the number of containers $\mathbf{C}$ (each determines the resources provided by an edge node), and effective time slot τ are recorded either in the auction smart contract (e.g., by a smart contract event) or in a new smart contract that can track the resource usage on the run. The client of each EIP and SP will monitor the resource contract record in the smart contract to negotiate the resource allocation and gather the payment or provision the tasks.

IV. EVALUATION

In this section, we present the experimental evaluation of the proposed smart contract-based resource allocation implemented and deployed on the real testbed, Rinkeby [22].

A. Setup

In the experiment evaluation, we focus on testing the performance of the algorithms when they are implemented in a smart contract and deployed on the real testbed, Rinkeby.

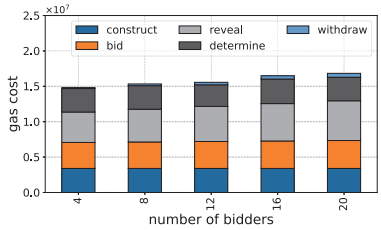

Fig. 6. Gas cost of different number of bidders

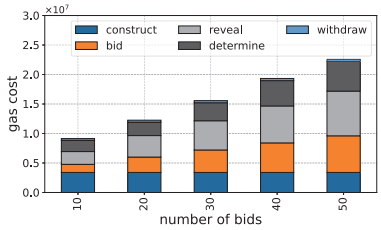

Fig. 7. Gas cost of different number of bids

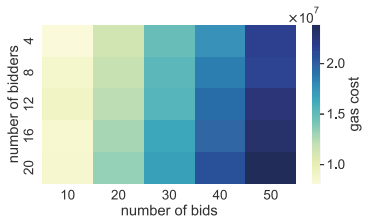

Fig. 8. Total gas cost comparison of different number of bids and different number of bidders

We assume that in each region, multiple EIPs participate as sellers, multiple SPs participate as buyers and a coordinator handles the coordination. For each auction, we keep the default setting as shown in Table II. We estimate the power

TABLE II
DEFAULT EXPERIMENT SETTING FOR EACH AUCTION

# of EIPs	6	# of bids (total)	30
# of SPs	6	# of containers per bid	100
PUE	1.2	electricity cost/operating cost	10%
gas price	20 gwei	gas limit (per block)	30M

consumption using a real server model that has the same performance as that of the IBM server x3550 (2 x [Xeon X5675 3067 MHz, 6 cores], 16GB) [23]. Each server hosts up to 5 service containers at a given time. The electricity price is generated based on the hourly real-time electricity price from NationalGrid's dataset [24]. We use the distribution of the data in 2015 from NationalGrid's hourly electricity price to simulate the fluctuation of the real electricity market. We also set the energy cost to 10% of the overall operating cost [25] and the Power Usage Effectiveness (PUE) is 1.2. For the utility model of the service provider, we choose a base rate similar to that of the a1.large instance (2 vCPUs and 4 GB memory) of Amazon EC2, which is $0.05 per hour usage. The linear growth of utility is based on the utility gain of the latency improvement which is in the 1-100 range.

B. Methodology

In our experiments, we evaluate two kinds of performance: (i) the performance of the smart contract which includes the gas cost of different scenarios and participants and function calls, (ii) the performance of the auction algorithms in comparison to other baselines. The evaluation metrics are defined as follows:

Gas cost: is the measurement of the cost of smart contracts. It is measured by the EVM which executes the function and each assembly operation (opcode) has a fixed gas cost based on its expected execution time.

Social Welfare: is a metric used to evaluate the performance of the auction. The social welfare can be calculated as the sum of the true valuation of the winners. The social welfare measures the efficiency of the auction. It is maximized if the goods are allocated to the buyers who value them the most.

Subsidy: is the difference between the payment from the buyers and the payment given to the sellers. The subsidy is generated based on the auction algorithms. As discussed in the definition, when it is negative, the auctioneer can gather fees from the auction, and when it is positive, the auctioneer needs to subsidize the trade to make up the difference.

C. Smart Contract Performance

As shown in Figure 6, we evaluate the gas cost with different number of bidders (sellers and buyers). The default number of bids is 30 as shown in Table II and the bids are evenly distributed to each bidder using a simple round robin algorithm. We illustrate the breakdown of the gas cost in five phases: (i) ***construct***, in which the coordinator creates the smart contract, (ii) ***bid***, in which the participants bid, (iii) ***reveal***, in which the participants reveal the sealed bid, (iv) ***determine***, in which the smart contract determines the winning bids by the auction algorithm, and (v) ***withdraw***, in which the coordinator and participants withdraw their funds. As shown in the results, we can see that when there are more participants, the overall gas cost has only a small increase and it only influences the *reveal* and *withdraw* phases as it increases the number of reveal and withdraw function calls. In Figure 7, the impact of the number of bids is evaluated. The setup is similar to the above experiment and we fixed the number of bidders to 12. We can see that with increasing the number of bids, the gas cost increases significantly especially for the *bid*, *reveal* and *determine* phases. The reason is that the number of bids influence the time complexity of each function call in the three phases. The influence of the number of bidders and bids are illustrated in Figure 8. We can get similar conclusion that the overall number of bids influences the gas cost much more than the number of bidders.

We also evaluate the gas cost for different participants and function calls in Figure 9 and 10. As shown in Figure 9, we can see that most of the gas cost is paid by the coordinator and the participants only pay gas cost when they bid or reveal the bids. As we design the auction mechanism based on McAfee mechanism, the coordinator has the opportunity to get payment from the auction and the participants can also pay management fees to the coordinator to cover such cost. In Figure 10, we observe similar results that show that *construct* and *determine* phases cost most of the gas. As shown in Table III, we convert the gas cost to real ether cost using the price listed in July 2021 (1 ether=$1787). As we can see, the coordinator needs to pay nearly 240 dollars to complete one auction, which is relatively high for the current market but there are many alternatives that can decrease the cost. For example, the coordinators can

TABLE III
A BREAKDOWN OF THE COSTS IN $ OF THE FUNCTION CALLS USING THE CONVERSION RATE OF 1 ETHER=$1787 AND THE GAS PRICE OF 20 GWEI (1 GWEI = 10^{-9} ETHER) AS LISTED IN JULY 2021

phase	gas cost						cost in $
	mean	min	25%	50%	75%	max	mean
construct	3400175	3400175	3400175	3400175	3400175	3400175	121.52
bid	126469	120002	120026	120056	137126	137156	4.52
reveal	433973	207976	218673	355149	491625	1873449	15.51
determine	3284284	1457251	2371536	3318685	4291935	5273596	117.38
withdraw	26899	23458	23458	28465	28465	28465	0.96

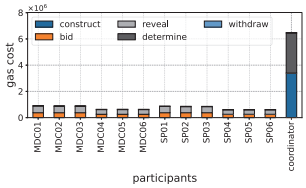

Fig. 9. Gas cost of different participants

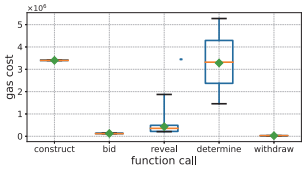

Fig. 10. Gas cost distribution of each function call

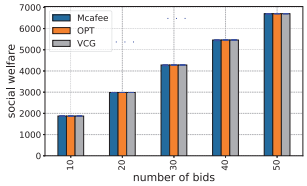

Fig. 11. Social welfare of different methods with different number of bids

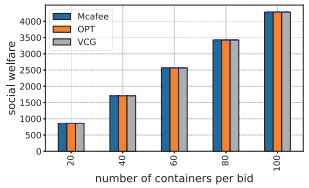

Fig. 12. Social welfare of different methods with different number of containers per bid

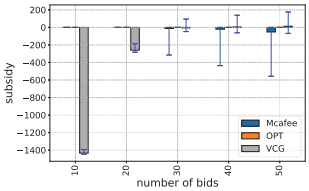

Fig. 13. Subsidy of different methods with different number of bids

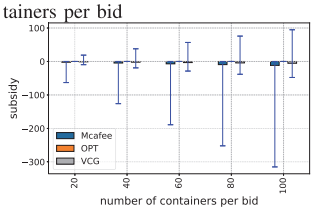

Fig. 14. Subsidy of different methods with different number of containers per bid

build a private blockchain (e.g., by using Hyperledger Fabric or Ethereum 2.0) to decrease the operating cost of running the smart contract. Each bidder may need to pay $4.5 for one bid and $15.5 for revealing all the bids (linear to the number of bids) on average. The withdrawal only needs less than $1.

D. Auction Performance

In this experiment, we test the performance of different auction algorithms, in terms of social welfare and subsidy. We compare the following auction algorithms:

McAfee [19]: is the auction mechanism we use in our method. It guarantees both truthfulness and weak balanced budget at the same time.

OPT: is a straightforward auction mechanism that always chooses the highest buy bids and lowest sell bids to trade. It is named as optimal single price omniscient (OPT) [26]. It can always guarantee balanced budge but not truthfulness.

VCG: is a well-known auction mechanism [27] that guarantees truthfulness but not balanced budget.

In Figure 11, we evaluate the social welfare of the above three auction algorithms with a different number of bids. We can see that all of the three methods have similar social welfare in different setups. From the theoretic aspect, only Mcafee may lose social welfare in the second condition (as shown in Algorithm 1) and because the bids are evenly distributed, the probability of the occurrence of the second condition is low. The result is similar when we increase the number of containers being traded in each bid as shown in Figure 12. When the subsidy is considered in the evaluation (Figure 13 and 14), we can see that VCG will suffer from positive subsidy and the coordinator needs to subsidize the trade. However, McAfee mechanism has weak balanced budget and the subsidy can be only negative. It means that this can be one possible way for the coordinator to gather fees and mitigate the operating cost of running the smart contracts. OPT always has balanced budget and the subsidy is always zero.

V. RELATED WORKS

Edge Computing has gained a lot of attention from the system community in the recent years. Resource allocation and management is a fundamental and important aspect in edge computing domain. Do et al. [28] propose a system for allocating fog computing resources to minimize the carbon footprint. Wang et al. [29] propose a mechanism to allocate the edge network and computing resources based on a deep reinforcement learning method. However, the above efforts do not consider the resource sharing problem between the different entities and they may suffer from inefficient resource usage due to not considering the resources from other entities. To solve the problems above, there have been solutions based on mechanism design (auction) to design resource sharing protocols between different entities. Xu et al. [8] proposed a framework to support sharing of edge devices among multiple communities by incentivizing the utility gain of the participants, which contains the results to complement the system performance aspects (latency, utilization, etc.) of the current work. However, the centralized auction handled by the infrastructure provider may lead to trust concern for the participants. Sun et al. [30] proposed two double auction algorithms to determine the matched pairs between the IoT devices and edge servers. Here, the usage of the above auction-based methods may be limited by their centralized trust assumption which requires all the participants to trust the centralized auctioneer. There have also been several efforts on decentralized auction-based mechanisms to allocate the edge

resources. Zavodovski et al. [31] proposed DeCloud which uses blockchain for managing the auction mechanism of edge resource usage using a weighted matching mechanism. Lin et al. [32] proposed a hierarchical real-time auction mechanism for allocating the resources. The real-time auction described in this work may not possible to be deployed in the public blockchain network and it may consume high transaction fees. In contrast to these related efforts, the proposed work develops a decentralized resource allocation platform, which enables long-term resource sharing between the ECIPs and SPs to increase their utilities based on a smart contract enabled double auction mechanism.

VI. Conclusion

In this paper, we propose a blockchain-based auction for allocating computing resources in an edge computing platform that allows service providers to establish resource sharing contracts with edge infrastructure providers using smart contracts in Ethereum. The proposed auction protocol decentralizes the trust using the blockchain network and reduces the trust concerns of centralized auction and the risks associated with a single point of failure. We implement a prototype of the proposed model on a real blockchain test-bed and our extensive experiments demonstrate the effectiveness, scalability and performance efficiency of the proposed approach.

VII. Acknowledgement

This work is partially supported by an IBM Faculty award for Balaji Palanisamy. Qingyang Wang acknowledges the partial support from the NSF CISE's CNS-2000681 grant.

References

[1] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014, pp. 169–186.

[2] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*. IEEE, 2014, pp. 1–9.

[3] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815*, 2015.

[4] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014.

[5] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*. IEEE, 2016, pp. 1–8.

[6] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[7] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[8] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *2017 IEEE international conference on edge computing (EDGE)*. IEEE, 2017, pp. 47–54.

[9] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[10] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[11] Amazon AWS, "AWS IoT Greengrass," https://aws.amazon.com/greengrass/, accessed Oct. 14, 2021.

[12] Amazon, "AWS Lambda," https://aws.amazon.com/lambda/, accessed Oct. 14, 2021.

[13] Azure, "Azure iot edge," https://azure.microsoft.com/en-us/services/iot-edge/, accessed Oct. 14, 2021.

[14] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[15] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures–a technology review," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, 2015, pp. 379–386.

[16] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *Open Systems (ICOS), 2015 IEEE Confernece on*. IEEE, 2015, pp. 130–135.

[17] M. Kaleem, A. Mavridou, and A. Laszka, "Vyper: A security comparison with solidity based on common vulnerabilities," in *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2020, pp. 107–111.

[18] R. B. Myerson and M. A. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of economic theory*, vol. 29, no. 2, pp. 265–281, 1983.

[19] R. P. McAfee, "A dominant strategy double auction," *Journal of economic Theory*, vol. 56, no. 2, pp. 434–450, 1992.

[20] S. K. Ong and A. Y. C. Nee, *Virtual and augmented reality applications in manufacturing*. Springer Science & Business Media, 2013.

[21] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[22] Rinkeby Ethereum Testnet, "Rinkeby: Ethereum Testnet," https://www.rinkeby.io/#stats, accessed Nov. 14, 2021.

[23] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[24] National Grid, "Hourly electric supply charges," https://www9.nationalgridus.com/niagaramohawk/business/rates/5_hour_charge.asp, accessed Sept. 22, 2021.

[25] J. Koomey, K. Brill, P. Turner, J. Stanley, and B. Taylor, "A simple model for determining true total cost of ownership for data centers," *Uptime Institute White Paper, Version*, vol. 2, p. 2007, 2007.

[26] A. V. Goldberg and J. D. Hartline, "Competitiveness via consensus." in *SODA*, vol. 3, 2003, pp. 215–222.

[27] T. Roughgarden, "Algorithmic game theory," *Communications of the ACM*, vol. 53, no. 7, pp. 78–86, 2010.

[28] C. T. Do, N. H. Tran, C. Pham, M. G. R. Alam, J. H. Son, and C. S. Hong, "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing," in *Information Networking (ICOIN), 2015 International Conference on*. IEEE, 2015, pp. 324–329.

[29] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on emerging topics in computing*, 2019.

[30] W. Sun, J. Liu, Y. Yue, and H. Zhang, "Double auction-based resource allocation for mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4692–4701, 2018.

[31] A. Zavodovski, S. Bayhan, N. Mohan, P. Zhou, W. Wong, and J. Kangasharju, "Decloud: Truthful decentralized double auction for edge clouds," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 2157–2167.

[32] H. Lin, Z. Yang, Z. Hong, S. Li, and W. Chen, "Smart contract-based hierarchical auction mechanism for edge computing in blockchain-empowered iot," in *2020 IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2020, pp. 147–156.