DeCloud: Truthful Decentralized Double Auction for Edge Clouds

Aleksandr Zavodovski[†], Suzan Bayhan[†‡], Nitinder Mohan[†], Pengyuan Zhou[†], Walter Wong[†], Jussi Kangasharju[†]
[†]University of Helsinki, Finland, {first name, last name}@helsinki.fi
[‡] TU Berlin, Germany, suzan.bayhan@tu-berlin.de

***Abstract*—The sharing economy has made great inroads with services like Uber or Airbnb enabling people to share their unused resources with those needing them. The computing world, however, despite its abundance of excess computational resources has remained largely unaffected by this trend, save for few examples like SETI@home. We present DeCloud, a decentralized market framework bringing the sharing economy to on-demand computing where the offering of pay-as-you-go services will not be limited to large companies, but ad hoc clouds can be spontaneously formed on the edge of the network. We design incentive compatible double auction mechanism targeted specifically for distributed ledger trust model instead of relying on third-party auctioneer. DeCloud incorporates innovative matching heuristic capable of coping with the level of heterogeneity inherent for large-scale open systems. Evaluating DeCloud on Google cluster-usage data, we demonstrate that the system has a near-optimal performance from an economic point of view, additionally enhanced by the flexibility of matching.**

I. Introduction

The sharing economy has made great inroads, as the examples of services like Uber or Airbnb demonstrate. They act as middlemen enabling people to share their resources (e.g., car, house) with the others. While the sharing economy has made a significant impact with real-world resources, provisioning of compute-on-demand resources, however, has remained an oligopoly of a few large companies. We believe that the time is now ripe for what might be called *decloudification*: creating efficient mechanisms for wide-scale sharing of computational resources (both managed and crowdsourced) between a wider array of providers and customers. We base this claim on the following observations. First, cloud computing is shifting toward edge computing, wherein cloud compute servers are augmented and assisted by additional resources located in the proximity of end-users [1], [2]. This trend is partly driven by new applications, such as Internet of Things (IoT) generating massive amounts of data, which require aggregation or filtering at the edge before being forwarded to a remote cloud. Many other applications, especially augmented reality (AR), have very stringent latency requirements, thus necessitating local processing. Second, in addition to the edge computing facilities installed by the operators, user-owned devices (computers, laptops, even phones) house significant compute capability which remains mostly underutilized [3], [4]. This capacity is located at the edge of the network, exactly where the new applications would require and could benefit from it. Finally, there is a trend towards establishing an open infrastructure for edge computing, where any motivated entity complying to common standards and practices can become an edge service provider (ESP), offering its computational capacity for running third-party applications close to the end-users and data producers, e.g., [5]–[7].

Our observations are backed by the emergence of new platforms, such as iExec [8], Sonm [9], and Golem [10]. They provide crowdsourcing capabilities without centralized brokerage, relying on smart contract [11] as a means to establish an agreement and to secure financial transaction. However, the design of an efficient market for decentralized open infrastructures remains largely unaddressed, and in this paper, we are motivated to tackle this challenge. We present *DeCloud*, a fully decentralized computational resource sharing framework backed by dominant strategy incentive compatible (DSIC) double auction [12], [13]. Incentive compatible auctions are also called *truthful* since both buyers and sellers are motivated to expose their true valuations and costs of goods to the auctioneer as sealed bids. Being a dominant strategy, truthful bidding leads to a maximum utility (payoff) for participants, greatly simplifying the behavior of the trading parties and reducing chances for market manipulation. However, in a distributed environment where an auctioneer is not present as a single entity, running a DSIC auction presents a certain challenge (mainly because of sealed bids), that we address with our proposed *two-phase bid exposure protocol*.

Fig. 1 gives an intuitive overview of DeCloud. There are two major stakeholders in the system, *clients* requesting computational resources, and *providers*, offering them. Technically, by submitting a request client expresses a wish that someone possessing hardware with certain characteristics will run client's container or VM[1] for an agreed period of time, whereas provider by posting an offer undertakes to execute such containers. DeCloud computes a near-optimal match (regarding economics and computational resources) between requests and offers. The market is built on top of a distributed ledger supported by smart contracts, which provides an execution environment for our auction mechanism. To protect clients from potentially malicious providers, DeCloud leverages recent trends, namely, trusted execution environments (TEEs), which make opportunistic ad hoc edge clouds feasible to implement securely on a large scale.

[1]For convenience, in the rest of this paper, we refer to containers as the units that clients submit for execution since they are more lightweight and became de facto standard in the applications based on the microservices architecture.

- accepted to IEEE ICDCS 2019

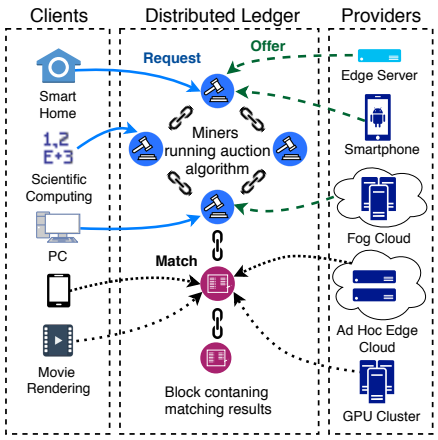


Fig. 1: DeCloud overview.

In this paper, we make the following contributions: i) Development of DSIC double auction mechanism with a bidding language explicitly designed for matching highly heterogeneous resources with diverse requests. ii) Addressing practical issues of running auction with sealed bids on the blockchain by our two-phase bid exposure protocol, iii) evaluation of truthful mechanism design impact on the economic performance, getting from 70% to over 85% of optimal welfare, depending on the market situation.

The rest of this paper is organized as follows. Section II presents the overview and motivation behind DeCloud. Section III describes the operation of DeCloud. Section IV concentrates on a description of theoretical model of the market we implement. Section V evaluates economic performance of DeCloud. In Section VI, we discuss alternative design choices and the impact of near-future technologies. The related work is the topic of Section VII, and Section VIII concludes the paper.

II. DeCloud Overview

Fig. 1 shows the *participants* of the system, clients and providers, interacting via distributed ledger. The latter is the central component of the system, responsible for the auction algorithm execution and storing the results, and next, we give a brief introduction to the underlying technology, focusing on blockchain enhanced by smart contracts.

A. Distributed Ledger in DeCloud

Basically, a blockchain is an ordered sequence of records, which are referred to as blocks. Each block contains a reference to the previous one, proof-of-work (PoW), timestamp and usually transaction information as a useful payload. The content of the block is cryptographically secured by PoW making blockchains secure by design. In practice, blockchain exists as a peer-to-peer network of participants, called miners, which take responsibility for generation and validation of new blocks according to a shared protocol standard. The security of a blockchain may be compromised only if the majority of network enters collusion; hence, blockchains achieve decentralized consensus and are considered to have high Byzantine fault tolerance [14]. One of the best-known examples of distributed ledger based on blockchain[2] is Bitcoin [16].

With Ethereum [17], blockchain was extended with working implementation of *smart contracts*, conceived much earlier by Nick Szabo [11] as "building blocks for digital economy". Smart contracts provide secure code execution on blockchain network and are ideal for the creation of dynamic online markets without a central authority. They expose *methods* like any program, which may be invoked by submitting transactions to the network. A miner receiving the call executes the code of contract and records changes to the blockchain state caused by the execution. The execution is verified by the rest of the miner network, and only if it is correct, the state changes are globally accepted.

The auction algorithm in DeCloud is executed by miners and collectively verified similarly to the code execution of smart contracts. The allocation results, i.e., computed mappings between clients and providers are stored in cryptographically secured blocks along with payment transactions, forming the blockchain. Participants submit bids in the same way as signed transactions are sent in any other blockchain-based protocol. There are public blockchains where any participant can join, and private, where only authorized miners are accepted. In DeCloud, both scenarios are viable: some mid-scale or even large cloud providers can have private blockchains, trading in DeCloud to balance the load and optimize machine running costs; general consumers can run the system publicly.

The main challenge of running DeCloud on top of distributed ledger is the openness of the current protocols supporting smart contracts. In such systems, the content of all transactions is public, and therefore visible to anyone connected to the overlay network as a miner or just an observer. One of the conditions to ensure truthfulness is that in DeCloud participants must not see each other bids before the allocation is computed. Submitting offers and requests as sealed bids prevent clients from manipulating the allocation result by tweaking their requirements and obsoletes strategies typical for ascending auctions. Motivated by the above considerations, we devise our own *two-phase bid expose protocol* and detail it in Section III.

Blockchain related technologies are rapidly developing, and there are upcoming systems promising anonymity of participants along with other improvements, such as low energy consumption and high transaction throughput. We discuss the relevance and impact of those near-future innovations on DeCloud in Section VI.

B. The Market Design

In DeCloud, both buyers and sellers submit their requests and offers to the market, specifying valuations and costs, respectively. Such market organization is called double auction, and McAfee [18] has shown that for double auction with more that one buyer and seller incentive compatible (truthful)

[2]Distributed ledger is not necessarily blockchain-based, e.g., there are cryptotokens such as Ripple [15] that use different technical solution.

mechanism exists. Moreover, truthful bidding is the dominant strategy, so McAfee's mechanism is DSIC; therefore rational participants get the best utility bidding their privately known valuations and costs. Truthfulness reduces chances for pricing manipulation and eliminates the need to develop complex bidding strategies. McAfee's mechanism assumes single type of goods and equal number of buyers and sellers. Our setting is much more complicated: we do not have discrete types of goods since the number of possible hardware configurations providers can offer is practically unlimited. We also do not expect one-to-one matchings between participants to happen very often. Given the above considerations, in Section IV we develop our own DSIC mechanism developing the ideas of McAfee's work further and give the formal definitions of the concepts presented above.

From an economic perspective, our mechanism approximates *social welfare* maximization, which is equivalent to giving the goods to the buyers who value them most, buying those goods from the sellers which have the lowest prices. However, the maximum welfare in DSIC double auction is not principally achievable as a consequence of trade reduction procedure needed to guarantee DSIC property. We devise an innovative technique to minimize welfare losses, and as our evaluation in Section V shows, we obtain from 70% to over 85% of the optimal welfare.

C. Extensible Bidding Language

The resources in DeCloud are much more heterogeneous than in usual cloud computing. Providers can offer an exuberant variety of hardware configurations, and there might be clients interested in a very particular setup. Therefore, we avoid the well-known approach of grouping physical machines (PMs) into categories (e.g., by performance) but instead develop a bidding language flexible enough to express needs (or capabilities) of the participants precisely. Described in detail, requests might have sometimes relatively low chances to find an exact match, and we solve this problem by devising a matching mechanism which while achieving high quality of the match, allows the clients to assert diverse kinds of tradeoffs they are willing to take. We enhance the expressive power of our bidding language by treating generic properties essential for edge computing, such as network latency or physical location, also as a specific resource. We describe the details in Section IV-B.

D. Privacy of the Client

In cloud and crowdsourced systems [10], [19], providers of PMs have been protected from malicious clients by isolated environments like virtual machines (VMs) or containerization software such as Docker [20]. However, a client's code and confidential data are potentially accessible to malicious providers with root level access to PMs. Large cloud providers have contracts with customers, but in a crowdsourced system, legal agreements cannot be assumed. Moreover, sensitive private data might be accessible by a potentially malicious provider.

DeCloud leverages trusted execution environment (TEE) to ensure client's privacy and security requirements. TEEs have become practical since the introduction of Software Guard eXtensions (SGX) by Intel [21] and have been adopted by ARM and AMD processors offering similar functionality as TrustZone technology [22]. SGX provides full hardware-based isolation in encrypted enclaves, which even a local operating system cannot access. There have been examples of Docker containers running securely inside SGX enclaves, e.g., SCONE [23]. Clients with lenient privacy requirements can use DeCloud without resorting to TEEs.

E. System Applicability

In this paper, we focus on matching computational demands in cloud and edge computing scenarios. As discussed above, this is an important and active area in the near future, and we consider it to be an excellent example of what DeCloud can do. Any cloud provider, from large to small, can join DeCloud and benefit from its optimized allocation of resources.

DeCloud is not restricted to cloud computing, and as a decentralized framework, it has the aptitude to be extended for matching other kinds of supply and demand in different walks of business. Many large companies, such as Uber or Airbnb, are based on the business model of acting as a trusted intermediary between suppliers (e.g., drivers or homeowners) and clients (e.g., passengers or travelers) in what are essentially using crowdsourced resources. There is a potential for systems like DeCloud to take over the role of the trusted intermediary in these kinds of services, removing the entity in the middle.

III. Two-Phase Bid Exposure Protocol

We describe the operation of DeCloud as a two-phase bid exposure protocol, with the main objective to ensure sealed bids while utilizing a transparent distributed ledger. It is worth noting that it is possible to implement DeCloud as a smart contract and run it on top of existing systems, such as [24] or [25]. The main reason for describing DeCloud as a separate protocol is the clarity and preserving the generality at a sufficiently high level.

Fig. 2 shows the workflow of the protocol. There is a participant (either a client or a provider), miner *A* that gets a block, and the rest of the miners on a blockchain network. The green line in Fig. 2 separates the flow into two phases: the bidding phase with sealed bids and the execution phase of allocation computation and agreement.

A. Sealed Bids

The participants send bids to the network signed by their private keys. Bids are submitted in the same way as transactions are posted to any other blockchain system, with one exception. To ensure non-disclosure, participants encrypt them entirely with temporary keys prior to submission.

The miner that gets the block computes the PoW solution for the block. At this moment, the block consists of encrypted data chunks. Upon generating the PoW solution, the miner shares the block with the rest of the network. The shared part

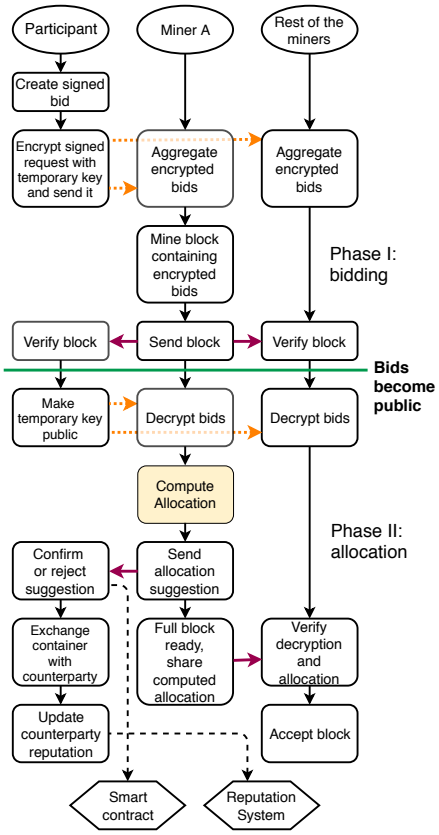


Fig. 2: Two-phase bid exposure protocol: interaction between participants, miners, and the matching algorithm.

is only the first part of an actual block, containing a reference to the previous block, PoW solution, and encrypted bids. We call it the *preamble* of the block. Participants and other miners receive the preamble and validate the PoW solution. If a valid preamble contains their bid, participants broadcast their temporary keys to the network. Now, the bids are disclosed, and the set of participants to be matched in the current round is known.

B. Allocation and Agreement

In the allocation phase, the miner decrypts the contents of the block and computes the allocation matching providers with clients. When the allocation is ready, the miner sends an *allocation suggestion* to the rest of the network. The allocation suggestion together with the set of temporary keys forms the *body* of a block. Upon receiving the block body, miners can compare the set of temporary keys with their own and conclude whether the miner had excluded anyone from the allocation. They also verify the accuracy of the allocation algorithm execution, and in the case everything is correct, miners accept the block.

The clients can accept or reject suggested allocation. This step is necessary because the clients do not choose the providers and resources, but instead, they are given the best possible match. There is a reputational penalty for successive rejections of matches. We choose not to give the providers the possibility to reject clients, although they may set a

TABLE I: Notation

Symbol	Description	Symbol	Description
$i \in N$	set of clients	$j \in M$	set of providers
$\mu \in \Upsilon$	set of miners	$\beta \in B$	set of blocks
$t \in T$	arbitrary time	$k \in K$	resource type
$t_r \in T$	timestamp of request	$t_o \in T$	timestamp of offer
$K_r \in K$	resource types of request r	$K_o \in K$	resource types of offer o
X_β	allocation in block β	w_β	welfare in block β
$r \in R_i$	request of client i posted at time t		
$o \in O_j$	offer of provider j posted at time t		
R_i^β	all requests of client i accepted in block β		
O_j^β	all offers of provider j accepted in block β		
v_r	true valuation of client i for its request r		
c_o	true cost of provider j for its offer o		
b_r	bid or reported valuation of client i for its request r		
b_o	bid or reported cost of provider j for its offer o		
p	clearing price		
p_r	payment of the client for request r		
π_o	revenue provider j receives from o		
u_r (u_o)	utility of the client i (provider j) for its request r (offer o)		
$x_{(r,o)}^\beta$	indicates whether r and o are matched in block β		
$\varphi_{(r,o)}^\beta$	fraction of o allocated to r in block β		
$K_{(r,o)}$	common resources of o and r, $K_r \cap K_o$		
t_r^- (t_r^+)	start (end) time of r		
t_o^- (t_o^+)	start (end) time of o		
ℓ_r(ℓ_o)	location parameter of request r (offer o)		
d_r	duration for which container of r must be up and running		
$\rho_{(r,k)}$	amount of the resource of type k that the request r needs		
$\sigma_{(r,k)}$	significance (or weight) of the resource of type k for the request r		
$\rho_{(o,k)}$	amount of the resource of type k that the offer o contains		

threshold for the reputation of the clients that they accept. Participants enter the agreement using a smart contract. This gives additional flexibility as these contracts can be executed on the same or a different blockchain system.[3] Clients respond by calling *accept* method of a smart contract and initiate the agreement with the provider. The smart contract checks that the allocation was generated, it is contained in the block that client references, and client's ID is associated with the particular provider. The smart contract enters the phase of *agreement*. A client not willing to accept an allocation calls the *deny* method of the contract. Such action will notify the provider that it needs to resubmit its offer to the network. Participants, whose bids were refused, can resubmit their bids.

IV. The Double Auction in DeCloud

We devise a double auction mechanism $\mathcal{A}$, which is: DSIC, strongly budget balanced (BB), and individually rational (IR). Formally, there are N clients and M providers. We use index $i \in [1, N]$ for clients, and $j \in [1, M]$ for providers. A client submits a request r at time t which goes into block β. Request r stands for single container client i needs to run. We define a request r of client i as follows:

$$r := < t_r, [\rho_{(r,k)}], [\sigma_{(r,k)}], t_r^-, t_r^+, d_r, b_r, \ell_r >, \qquad (1)$$

where $\rho_{(r,k)}$ is the needed resource of type $k \in K$, and $\sigma_{(r,k)}$ is the significance of that resource (see Section IV-B). The time period for when the client needs the resources is given by t_r^- and t_r^+. It is not necessary for the container to run

[3]Smart contracts cannot access mutable off-chain data because the execution must be deterministic. However, there are approaches removing that limitation, namely, the oracles, such as [26]. Additionally, interlegders, e.g., [27] promise to facilitate cross-chain communication.

from t_r^- to t_r^+; an additional parameter duration d_r specifies for how long the container must be continuously executed. In case $d_r = t_r^+ - t_r^-$, the container must be run from t_r^- to t_r^+. The client reports its bid b_r in its request; in DSIC auction it is equal to its private valuation v_r, e.g., monetary value, for this request. The request is tagged with the location ℓ_r, which is the location where the client would prefer to run its edge service. This might be either geo-location or a network address.

Likewise, provider submits its offer(s) o which represents a computational device with amount $\rho_{(o,k)}$ of the resource of type $k \in K_o$. A container can be matched with a single device and devices are capable of running multiple containers. Each offer o has associated t_o^- and t_o^+ parameters, indicating when the offered device is available. We define offer o as follows:

$$o :=< t_o, [\rho_{(o,k)}], t_o^-, t_o^+, b_o, \ell_o >, \tag{2}$$

where ℓ_o is the location of the provider and c_o is the cost of the provider corresponding to this offer. The provider publishes its bid b_o in its offer, which we later prove to be equal to its privately known cost c_o. In DSIC auction, $b_r = v_r$, and $b_o = c_o$, respectively.

There is set of miners $\mu \in \Upsilon$ that generates blocks $\beta \in B$. Let R^β and O^β denote all requests and offers accepted in block β. We define the matching between requests and offers by a binary matrix $X_\beta = [x^\beta_{(r,o)}]$ where $r \in R^\beta$ and $o \in O^\beta$ and $x^\beta_{(r,o)} = 1$ states that offer o and request r are matched. The allocation over the entire lifespan of the system is then: $X = \bigcup_{\beta \in B} X_\beta$. Table I summarizes our notation.

A. Welfare

Most common optimization targets for economic performance are revenue and welfare. Optimizing for revenue pursues the benefit of the seller, but as we want to motivate clients and providers equally, we opt for welfare optimization. For a double auction, welfare is the difference between the total value of the buyers and total cost of the sellers [28]. Each client i has a valuation v_r for its request r, and each provider j has a cost c_o for its offer o. If r and o are matched in the block β, the resulting welfare is $w_{(r,o)} = v_r - \varphi^\beta_{(r,o)} c_o$, where $\varphi^\beta_{(r,o)}$ is the resource fraction of o allocated to r. The total welfare of participants matched in block β equals to:

$$w_\beta(X_\beta) = \sum_{r \in R^\beta} \sum_{o \in O^\beta} v_r x^\beta_{(r,o)} - \sum_{o \in O^\beta} \sum_{r \in R^\beta} x^\beta_{(r,o)} \varphi^\beta_{(r,o)} c_o. \tag{3}$$

As each block is considered independently, we can formulate our problem as welfare-maximization for each block:

$$\max \sum_{r \in R^\beta} \sum_{o \in O^\beta} v_r x^\beta_{(r,o)} - \sum_{o \in O^\beta} \sum_{r \in R^\beta} x^\beta_{(r,o)} \varphi^\beta_{(r,o)} c_o \tag{4}$$

subject to:

$$\sum_{o \in O^\beta} x^\beta_{(r,o)} \leqslant 1 \ , \forall r \tag{5}$$

$$\varphi^\beta_{(r,o)} = \frac{d_r}{t_o^+ - t_o^-} \frac{1}{|K_{(r,o)}|} \sum_{k \in K_{(r,o)}} \frac{\rho_{(r,k)}}{\rho_{(o,k)}} \quad \forall r, \forall o \tag{6}$$

$$\sum_{r \in R^\beta} \varphi^\beta_{(r,o,k)} \leqslant 1, \ \forall o, \forall k \tag{7}$$

$$\rho_{(r,k)} x^\beta_{(r,o)} \leqslant \rho_{(o,k)}, \ \forall o, \forall r, \forall k \tag{8}$$

$$v_r \geqslant \varphi^\beta_{(r,o)} c_o, \ \forall r, \forall o \tag{9}$$

$$x^\beta_{(r,o)} t_o^- \leqslant x^\beta_{(r,o)} t_r^-, \ \forall r, \forall o \tag{10}$$

$$x^\beta_{(r,o)} t_o^+ \geqslant x^\beta_{(r,o)} t_r^+, \ \forall r, \forall o \tag{11}$$

$$v_r \geqslant 0, v_r \in \mathbb{R}, \ \forall r \tag{12}$$

$$c_o \geqslant 0, c_o \in \mathbb{R}, \ \forall o \tag{13}$$

$$x^\beta_{(r,o)} \in \{0, 1\} \tag{14}$$

Our objective (4) states the goal of our problem as maximization of the welfare for block β. Const. (5) ensures that a request is matched to at most one provider's bid while (6) defines the fraction of resources when a request r is assigned to offer o. Const. (7) is necessary for the feasibility of resource allocation: fraction of resources of each resource type allocated by each provider for its offer can be at most 1. Similarly, Const. (8) ensures that if o is allocated for r, o has sufficient quantity of resource ρ to serve r. Const. (9) ensures that the valuation of the client is greater than the cost of allocated resources. We ensure temporal constraints by (10) and (11). Valuation and cost can only be non-negative rational numbers (12) and (13). Finally, (14) indicates that our decision variables are binary integers. Since transactions are included in blocks nondeterministically, total welfare of the system is:

$$w(X) = \sum_{(X_\beta, \beta) \in (X, B)} w_\beta(X_\beta) \tag{15}$$

The task of optimizing welfare of the entire system is then reduced to finding an optimal allocation X^*_β for each block in terms of welfare:

$$X^* = \{\forall \beta, \ X_\beta \mid \arg\max_{X_\beta} w_\beta(X_\beta)\} \tag{16}$$

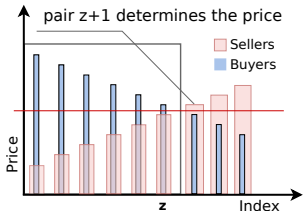
$$w^*(X^*) = \sum_{(X^*_\beta, \beta) \in (X^*, B)} w_\beta(X^*_\beta). \tag{17}$$

Since maximization of (17) will not render a DSIC mechanism, we use it for the evaluation of economic performance.

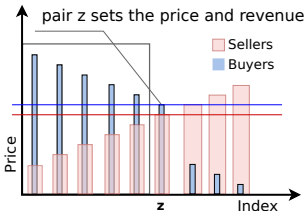
B. DeCloud Matching Heuristic

DeCloud exhibits a high degree of heterogeneity in supply and demand. We define a bidding language for clients to characterize required resources for their tasks and specify weights to indicate their importance. Let $\sigma_{(r,k)}$ represent the significance of a resource of type k for a client. The client may assert that the resource is strictly required by setting $\sigma_{(r,k)} = 1$, and $0 < \sigma_{(r,k)} < 1$ otherwise. A resource type k can represent a broad range of resources, e.g., latency, reputation, the presence of SGX.

Normally, a similarity measure like the dot product [29] could be used to determine the allocation, but it does not work well when clients can specify weights for their requests. We solve this by augmenting geometric distance with the concept

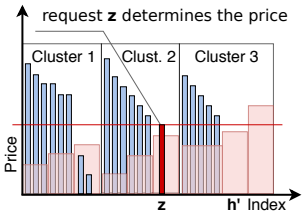


(a) Trading price is determined by pair z+1.

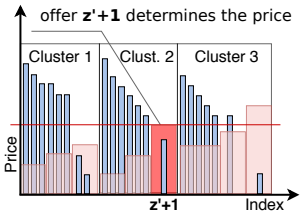


(b) Trade reduction: pair z is excluded.

Fig. 3: McAfee double auction matches low cost sellers with high paying buyers maximizing the welfare.



(a) Trading price is determined by request z.



(b) Trading price is determined by offer z'+1.

Fig. 4: DeCloud mechanism: a mini-auction containing three clusters. Requests and offers are matched within the borders of clusters, however, the price is common for all three clusters.

of field, where providers exert a gravity-like force. Formally, we define the *quality of match* between r and o as:

$$q_{(r,o)} = \sum_{k \in (K_r \cap K_o)} \sigma_{(r,k)} \frac{\rho'_{(o,k)}}{|\rho'_{(o,k)} - \rho'_{(r,k)}|^2 + 1} \tag{18}$$

where $\rho'_{(o,k)}$ and $\rho'_{(r,k)}$ are normalized $\rho_{(o,k)}$ and $\rho_{(r,k)}$. For normalization, we take the maximum value of the resource from offers or requests of the current block as a maximum of the scale and zero as a minimum. Using (18), we rank all offers for a particular request with at least one common resource, i.e., $|K_r \cap K_o| > 0$.

C. The Auction Mechanism

Before proceeding with the description of $\mathcal{A}$, it is convenient to look at McAfee's work [18], which presents DSIC mechanism for a double auction with more than one buyer and seller pair. In McAfee's auction, single units of identical goods are traded, so we use v as unit valuation and c as unit cost. The mechanism sorts buyers by their valuation in descending order and sellers in ascending order by cost. The last pair in which buyer has higher (or equal) valuation than seller's cost has the break-even index z. If there are more pairs than z and $p = (v_{z+1} + c_{z+1})/2 \in [c_z, v_z]$, then every participant trades at price p, see Fig. 3a. If the latter condition does not hold, buyers pay v_z, and sellers receive payment of c_z, as shown in Fig. 3b. To preserve DSIC property, the pair z has to be excluded from the trade, hence the procedure is referred to as *trade reduction*.

As it is clear from Fig. 3b, in the case of trade reduction McAfee's auction is not *strongly*[4] budget balanced (BB) since buyers pay more than sellers receive. This situation is desirable if the difference between payments and revenues is used as a reward of the auctioneer. However, in our setting, this is not the case because miners responsible for the algorithm execution are rewarded by cryptotokens emission. Fortunately, Segal-Halevi et al. [30] present strongly BB version of McAfee's mechanism achieved by slight modification of payment rule, which we also use and describe in more detail later.

Both [30] and [18] apply to the single type of goods, but goods are not discretized by type in DeCloud. Moreover, the characteristics of goods are not public in the bidding phase, so clients cannot target them explicitly. We solve this issue by resorting to the concept of *single-minded* [31] bidders. In the context of DeCloud, we consider clients as single-minded since they are interested in either winning the best approximation of the good specified in their bid (that they can also afford) or remaining unallocated at all.

Given the above considerations, we start computing the allocation by grouping together offers and requests according to quality of match heuristics (18). This grouping results in *clusters*, and each cluster contains set of offers and set of requests, such that any offer in the set satisfies certain quality of match criteria for each member of the cluster's request set.

Within a single cluster, offers and requests are likely to differ by their characteristics. Hence, for ranking them by their costs and valuations as in McAfee mechanism, we apply normalization to obtain the valuation/cost per unit of resource. Since offers and requests are for different time periods, we divide costs and valuations by corresponding timespans: $\frac{c_o}{t_o^+ - t_o^-}$ and $\frac{v_r}{d_r}$. To find the valuation and cost per unit resource, we define a set of common resource types $K_{\mathcal{CL}} = (\cup_{r \in \mathcal{CL}} K_r) \cap (\cup_{o \in \mathcal{CL}} K_o)$ in a cluster $\mathcal{CL}$ for all requests and offers. We compute the *virtual maximum* of the cluster by taking the maximum for each resource type that some $o \in \mathcal{CL}$ has: $\mathcal{M}_{\mathcal{CL}} = \{\forall k \in K_{\mathcal{CL}} \mid \rho_k = \max_o(\rho_{(o,k)})\}$. By treating each resource type as a vector component, we can compute a norm $\|\mathcal{M}_{\mathcal{CL}}\|_2$. Using the norm together with the magnitude of the resource vector of the offer ρ_o, we calculate the fraction of the virtual maximum that the offer corresponds to: $\nu_o = \frac{\|\rho_o\|_2}{\|\mathcal{M}_{\mathcal{CL}}\|_2}$. The normalized cost is $\hat{c}_o = \frac{c_o}{\nu_o(t_o^+ - t_o^-)}$, which is the cost of virtual maximum scaled down by time and up by a fraction of resources that the offer has.

For the requests we need to take into account critical resources, e.g., if a container uses 100% of CPU, no other request can be allocated to the offer at the same time irrespective of the amount of other available resources. Logically, the sender of such request should pay 100% of clearing price. We define the set of critical resource types $K_{\mathcal{CR}}$ such as CPU, memory, and disk, and add to $K_{\mathcal{CR}}$ resource types that are declared in all requests of $\mathcal{CL}$: $K_{\mathcal{CR}} = K_{\mathcal{CR}} \cup (\cap_{r \in \mathcal{CL}} K_r)$. Next, we find the maximal percentage of critical resource usage by request r as follows: $\nu_{\mathcal{CR}} = max(\forall k \in K_{\mathcal{CR}}$:

[4]In strongly BB auction sellers receive as much as buyers pay and there is no profit left for the auctioneer. McAfee's mechanism is BB since buyers pay more than sellers receive, but it is not strongly BB.

$\frac{\rho_{(r,k)}}{\rho_k \in \mathcal{M}_{\mathcal{CL}}}$). The resource fraction of $\mathcal{M}_{\mathcal{CL}}$ that request r uses is the greatest of critical resource usage and overall utilization: $\nu_r = max(\nu_{\mathcal{CR}}, \frac{\|\rho_r\|_2}{\|\mathcal{M}_{\mathcal{CL}}\|_2})$. Finally, normalized valuation equals $\hat{v}_r = \frac{v_r}{\nu_r d_r}$. To determine the payment for request r, we scale unit price:

$$p_r = \nu_r p. \tag{19}$$

To minimize the adverse effect of trade reduction and improve chances that there are more than a single pair of participants engaging in trade, we group clusters in *mini-auctions* according to their price range. Fig. 4 illustrates the idea of a mini-auction. In our setting, there is no 1-to-1 matching of buyers and sellers, but multiple buyers can be mapped to a single seller. We use separate indices z and z' for the last request and offer having $\hat{v}_z > \hat{c}_{z'}$. By price compatibility we mean that for clusters a and b the following conditions will hold: $\hat{v}_{z,a} > \hat{c}_{z',b}$ and $\hat{v}_{z,b} > \hat{c}_{z',a}$.

In determining the price, we follow [30], with two major cases, as shown in Fig. 4. Formally, the price is:

$$p = min(\hat{v}_z, \hat{c}_{z'+1}). \tag{20}$$

When there is no offer $z'+1$, we assume $\hat{c}_{z'+1} = \infty$. Request r_z or offer $o_{z'+1}$ determining the price is always excluded from trade since allocated participants cannot affect the price. Since we allow multiple requests (and offers) from a single client (provider), not only r_z and $o_{z'+1}$, but also all other requests belonging to the same client (provider) will be excluded from the same mini-auction.

D. Incentive Compatibility

Formally, the auction is incentive compatible (truthful) if utilities of the client and provider are maximized when participants bid their true valuation, i.e., $b_r = v_r$ and $b_o = c_o$. The utility of a client from an accepted request r is $u_r = v_r - p_r$, and $u_o = \pi_o - c_o$ of provider for offer o, where p_r is payment of a client, and π_o is revenue of the provider: $\pi_o = \sum_{r \in o} p_r$. If a participant is not allocated, we assume its utility to be zero and, consequently, payment is also zero in such case.

The proof of incentive compatibility mainly follows [18], [30]. The payment (revenue) rule is monotonic since increasing the reported valuation b_r (or decreasing reported cost b_o) only increases the chances of participants to get allocated. The determined price is also threshold: reporting a valuation below (or cost above) will lead to losing the allocation. The price never depends on the reported valuation or cost of allocated participants. Untruthful bidding has the following impact on client's utility:

1) Client with index smaller than z overbids: $b_r > v_r$. If $p_r > v_r$, then its utility will be negative ($u_r < 0$), as it wins at price exceeding v_r. If $p_r \leq v_r$, then utility stays the same as in the case of truthful bidding.
2) Client with index smaller than z underbids: $b_r < v_r$. If $p_r > b_r$, then the client's utility will be zero ($u_r = 0$), as it loses the auction. In case of truthful reporting, given $v_r > p$, the client's utility is greater than zero: $u_r > 0$. If $p_r \leq b_r$, utility remains unaffected.
3) For client with index z not determining the payment, the reasoning is similar as in previous two cases.
4) Client with index z determining the payment: underbidding will not improve the payoff since client will be excluded as if reporting truthfully. If client wins as a result of overbidding, there will be some client x with $b_x \geq v_r$, resulting in $u_r \leq 0$.
5) Client with index $> z$ losing the allocation: the reasoning is the same as in 4 above.

When bidding $b_r \neq v_r$ the buyer gets $u_r = v_r - p_r \geq b_r - p_r$, thus optimal strategy is to bid $b_r = v_r$. The reasoning for providers is symmetric.

However, due to not having 1-to-1 correspondence between requests and offers, there is a possibility to game the system. There is an offer h' in Fig. 4, that is not allocated because there are not enough requests. The provider would benefit from reporting $b_o < c_o$, because then h' might become the first offer in cluster 3 and thus get allocated, making $u_o > 0$. A similar strategy works for requests in case there are not enough offers to allocate all clients. When low on demand, the solution is to distribute existing requests evenly among offers and exclude redundant offers randomly. We also apply random exclusion of requests in case of a supply shortage. These measures lower welfare but are required to preserve incentive compatibility.

Cloud auctions are challenging for proving truthfulness, since not only economic parameters are subject to manipulation, but also requirements and submission times of buy and sell orders [32]. We show that participants cannot improve their utilities by reporting parameters untruthfully. As for submission times, t_r and t_o, we solve ranking ties by choosing request or offer with lower submission time, thus $t_r \leq t_{r'} \Rightarrow u_r \geq u_{r'}$ and $t_o \leq t_{o'} \Rightarrow u_o \geq u_{o'}$, and there is no incentive in delaying the submission. If any of the requirements, $[\rho_{(r,k)}], [\sigma_{(r,k)}], t_r^-, t_r^+, d_r, \ell_r$, is reported untruthfully in request r', then r' will be allocated to some offer o', resulting in $q_{(r,o')} \leq q_{(r,o)}$ by (18). Thus reporting r' is equivalent of bidding for the wrong good in another kind of an auction, so this contradicts rationality of the buyer. On the provider side, the hardware characteristics are reported by the system software. Participants can only decrease their utility by reporting untruthfully any part of an order; thus $\mathcal{A}$ is DSIC, assuming IR, that we define and prove next.

E. Individual Rationality

To be incentive compatible, our mechanism needs to satisfy IR, which means that buyers never pay more than they bid and sellers receive no less than they ask. According to (20) for clients we have two cases:

1) $\hat{v}_z < \hat{c}_{z'+1} \Rightarrow p = \hat{v}_z$. By (19), $p_r = \nu_r \hat{v}_z \leq \hat{v}_r = \nu_r v_r \Rightarrow p_r \leq v_r$, since $\hat{v}_z \leq \hat{v}_r$ and $\nu_r \leq 1$.
2) $\hat{v}_z > \hat{c}_{z'+1} \Rightarrow p = \hat{c}_{z'+1}$. The reasoning from the point above applies since the payment will be less than $\hat{v}_z$.

Providers are not guaranteed to be fully allocated. Thus we compute unit price based on the revenue they receive from the allocated fraction, and show that under this unit price their

Algorithm 1 Double auction allocation

Inputs: Offers O^β and requests R^β
Outputs: Matching between offers and requests
clusters $\leftarrow \{\varnothing\}$
for $r \in R^\beta$ **do**
  feasible $\leftarrow$ **filter** O^β **by** constraints of r
  feasible $\leftarrow$ **rank** *feasible* **by** $q_{(r,o)}$ **descending**
  $best_r \leftarrow$ **top from** *feasible*
  UPDATECLUSTERS(*clusters*, r, $best_r$)
for *cluster* $\in$ *clusters* **do**
  allocate r, $o \in$ *cluster* greedily
  determine $\hat{v}_z, \hat{c}_{z'+1}$ for *cluster*
auctions $\leftarrow$CREATEMINIAUCTIONS(*clusters*)
while *auctions* $\neq \varnothing$ **do**
  auction $\leftarrow$**top from** *auctions* **by** *welfare*
  PERFORMTRADEREDUCTION(*auction*)
  remove $r, o \in$ *auction* **from** $\forall a \in$ *auctions*
  remove *auction* **from** *auctions*
return $\{o \in O^\beta : o_{assigned} \neq \varnothing\}$

Algorithm 2 Update clusters procedure

procedure UPDATECLUSTERS(*clusters*, r, $best_r$)
  if $\nexists\, c \in$ *clusters* : $c_{offers} = best_r$ **then**
    clusters $\leftarrow$ {*clusters* $\cup$ <*offers*: $best_r$, *requests*: r>}
  subsets $\leftarrow \{c \in$ *clusters* $\mid c_{offers} \subseteq best_r\}$
  supersets $\leftarrow \{c \in$ *clusters* $\mid best_r \subseteq c_{offers}\}$
  for *subset* $\in$ *subsets* **do**
    $subset_{requests} \leftarrow \{subset_{requests} \cup r\}$
    for *superset* $\in$ *supersets* **do**
      $subset_{requests} \leftarrow \{subset_{requests} \cup superset_{requests}\}$
  for $c \in$ *clusters* : $c_{offers} \neq best_r$ **do**
    intersection $\leftarrow \{c_{offers} \cap best_r\}$
    if $|$*intersection*$| > 1$ **then**
      $x \leftarrow x \in$ *clusters* : x_{offers} = *intersection*
      if $x = \varnothing$ **then**
        clusters $\leftarrow$ {*clusters* $\cup$ **new** <*offers* : *intersection*, *requests* : $r \cup c_{requests}$>}
      else
        $x_{requests} \leftarrow \{ x_{requests} \cup r\}$

revenue would be at least as big as their cost in the case of full allocation and trading resources equal to $\mathcal{M}_{\mathcal{CL}}$ (assuming $\nu_z = \nu_o = 1$). There are same two cases as for clients:

1) $\hat{v}_z < \hat{c}_{z'+1} \Rightarrow p = \hat{v}_z = \nu_z v_z \geq \hat{c}_o = \nu_o c_o = c_o$, since $\hat{v}_z \geq \hat{c}_o$.
2) $\hat{v}_z > \hat{c}_{z'+1} \Rightarrow p = \hat{c}_{z'+1} = \nu_{z'+1} c_{z'+1} \geq \hat{c}_o = \nu_o c_o = c_o$, since $\hat{c}_{z'+1} \geq \hat{c}_o$.

We have shown that $\mathcal{A}$ is IR for both clients and providers.

F. Allocation Algorithm

We now describe our allocation algorithm, shown as Algorithm 1. We follow the steps defined in Section IV-C,

Algorithm 3 Create mini-auctions procedure

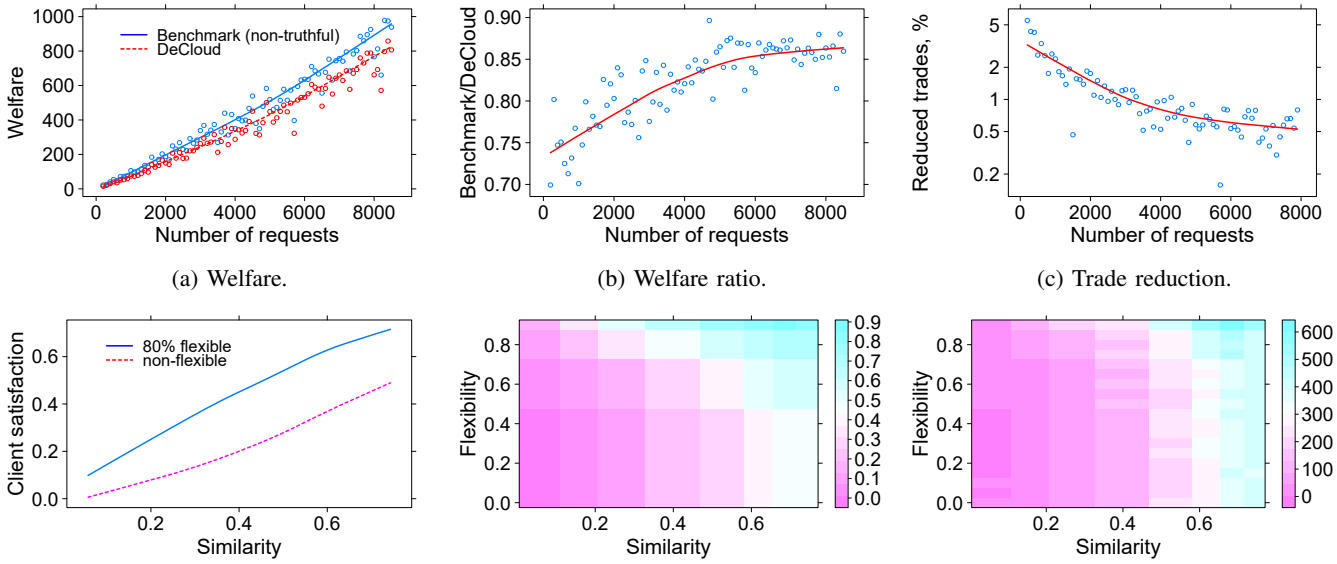
procedure CREATEMINIAUCTIONS(*clusters*)
  roots $\leftarrow$ find minimun nonoverlapping ranges of $[\hat{v}_z, \hat{c}_z]$ from $\forall$ *clusters*
  for $c \in$ *clusters* **do**
    for *root* $\in$ *roots* **do**
      if c not compatible with *root* **then**
        continue
      for *node* $\in preorder($*root*$)$ **do**
        if *node* is leaf and compatible(c,*node*) **then**
          append c to *node*
        if not compatible(c,*node*) **then**
          if compatible(c,parentof(*node*)) **then**
            append c to parentof(*node*)
  for *root* $\in$ *roots* **do**
    for *leaf* $\in postorder($*root*$)$ **do**
      yield path to root as miniauction

Algorithm 4 Perform trade reduction procedure

procedure PERFORMTRADEREDUCTION(*auction*)
  $p \leftarrow$ select minimun of $\hat{v}_z$, $\hat{c}_{z'+1}$ from $\forall$ *cluster* $\in$ *auction*
  if $p = \hat{v}_z$ **then**
    exclude requests of i, which submitted z
  else
    exclude offers of j, which submitted $z' + 1$
  for *cluster* $\in$ *auction* **do**
    if exist unallocated r, $o \in$*cluster*, having $\hat{v}_r \geq p$, $\hat{c}_o \leq p$ **then**
      randomize the allocation of *cluster*
  compute payments and revenues using price p

guaranteeing incentive compatibility and minimizing adverse effects of trade reduction. Despite the seeming complexity, the main task of the algorithm is to arrange requests and offers like in the pattern displayed in Fig. 4. First, we form clusters (see Algorithm 2). The purpose of clustering is to group together as many requests as possible with the same set of best offers according to our quality of match heuristics (18). While forming clusters, we also filter out offers incompatible by temporal or resource constraints, fit requests and offers together by matching greatest normalized valuation $\hat{v}_r$ with lowest cost $\hat{c}_o$, and determine z, z', and $z' + 1$ indices. To minimize the effect of trade reduction, we group clusters into mini-auctions (see Algorithm 3). Grouping is performed by building trees of clusters, where a child cluster is compatible by price with its parent. We use clusters with minimal incompatible price ranges as the roots of a forest; the problem to find those is equivalent of finding non-overlapping maximum weights intervals, which can be solved by dynamic programming in polynomial time.

After forming the mini-auctions, the price can be determined. In case there are excessive requests and offers, that have $\hat{v} > p$ and $\hat{c} < p$, we randomize the allocation to preserve truthfulness. We use a pseudorandom approach, using evidence



(a) Welfare. (b) Welfare ratio. (c) Trade reduction.

(d) Client satisfaction. (e) Client satisfaction heatmap. (f) Welfare heatmap.

Fig. 5: Performance of DeCloud against a non-truthful benchmark and the effect of flexibility on user satisfaction and welfare.

of a block as a random seed so that randomization is also verifiable. After the unit price is defined, the trade reduction procedure is performed in the Algorithm 4.

V. Performance Evaluation

First, we evaluate the performance of DeCloud by assessing how DSIC design affects the welfare of the system, as defined in (3). We use Google Cluster Data (CPU, RAM, and Disk) [33] to generate client requests realistically. For physical capabilities of providers (processing cores, memory, disk etc.) along with pricing data, we use data from Amazon EC2 M5 instance types [34]. We set providers' resources in a range between 2–16 CPU cores and 8–64 GB RAM. The valuation of each request is calculated as a cost of its best match offer multiplied by a random uniform coefficient in the range of [0.5, 2]. Our benchmark is a double auction using a similar algorithm, but without trade reduction and pseudorandomization, thus producing the best possible welfare under greedy allocation while being non-truthful. First, we consider a scenario with inflexible requests, i.e., the client gets always 100% of requested resources.

Figure 5a shows welfare for the benchmark and DeCloud, and Figure 5b – the ratio between them with the increasing number of requests. We also plot the Loess curves to show the trend in welfare and the welfare ratio. Even in the worst case, DeCloud achieves 75% of the welfare of the benchmark with up to 85% in larger systems. This slight decrease in welfare is the tradeoff we have to pay for a DSIC auction.

Figure 5c shows the percentage of reduced trades. Similarly to the case of welfare, DeCloud benefits from increasing market size, resulting in a smaller fraction of excluded trades. Due to a grouping of clusters into mini-auctions, the amount of excluded trades stays below 5%, dropping to 0.5% in large systems.

Second, we evaluate the effect of flexibility on the clients' satisfaction and welfare. For this evaluation, we generated sets of offers and requests distributions with various degrees of Kullback-Leibler divergence, e.g., when clients want mostly 8 cores CPUs, the majority of offered CPUs have only 2 cores, and so on. The similarity axis of Figures 5d, 5e, 5f is calculated as $1 - KLD(R^{\beta}, O^{\beta})$, where KLD is Kullback-Leibler divergence regarding resources. The *satisfaction* is defined as a fraction of allocated requests, one being the maximum. Figures 5d and 5e show the positive effect of flexible matching on the satisfaction of clients, while Figure 5f on the welfare. As Figure 5d suggests, 80% flexibility results in stably higher satisfaction, being a decent improvement.

The results are encouraging since even in the worst case, DeCloud only loses 5% of the trades while achieving 75% of the optimal welfare. In larger systems, reduced trades drop to 0.5%, and the welfare climbs to 85% of the non-truthful benchmark. Given that the key contribution of DeCloud is a truthful, decentralized auction, we consider these small performance tradeoffs acceptable, given the potential of DeCloud for enabling more efficient use of existing resources.

VI. Discussion

We now discuss practical issues related to DeCloud. Ethereum has been blamed for high costs of contracts execution. We believe that competition with other solutions, such as [25], will result in much lower costs. Running code verifiably on blockchain has been reasonably criticized for low throughput, but the problem is getting addressed in recently conceived systems, namely, Plasma [35] and TrueBit [36]. We consider running DeCloud as an Ethereum Plasma sidechain as a reasonable option.

Our system suffers from what is known as a *verifier dilemma*, pointed out in TrueBit. The problem arises when

miners do not have an incentive to validate proof-of-work because of its complexity. Fortunately, authors of TrueBit present a solution[5] that we may also incorporate. Another aspect of TrueBit is formal verification of computation which was performed by peers. In our case, such a verification appears to be overkill since DeCloud is designed to run mostly containerized service applications, whereas TrueBit – to provide scalability to smart contracts. Instead, we consider that either incorporating reputation system or linking DeCloud to an existing one [37]–[39] would be sufficient in most cases.

Systems such as Strain [40] for running an anonymous auctions on blockchains or even more general framework for crowdsourcing ZebraLancer [41] are of particular interest to us. By introducing our two-phase bid expose protocol (Section III) we are clearly on the accountability side in anonymity vs. accountability dispute. Our bias is motivated by: i) possibility to link participants to reputation systems, either built-in or third-party, ii) ability of clients and providers to infer their valuations from historical prices, iii) simplicity of implementation. Nevertheless, there are no technical restrictions for either running our auction mechanism on top of aforementioned systems or utilizing techniques from [42], [43] in the scenario emphasizing anonymity and secrecy of the historical data.

High energy consumption required by proof-of-work (PoW) has plagued the blockchain platforms. Fortunately, the numerous green solutions have been presented, of which, e.g., proof-of-stake [44] based Casper [45] will soon replace energy wasteful PoW of Ethereum. For us, systems like Sawtooth [46] are of particular interest, since they replace cryptographic computations of PoW by practically any useful computation.

Because of the workflow for processing the sealed bids and allocation correctness verification, the allocations will be computed in rounds which correspond to the generation of blocks by miners. Since the generation of blocks is non-deterministic, participants are practically agnostic of rounds and the system will have an online appearance to users (with some observed delay).

VII. Related Work

One of the earliest crowdsourced distributed computing platforms is BOINC [19], which was conceived as a generalization of SETI@home computing framework. BOINC remains highly specialized semi-centralized system, and it is not directly comparable with DeCloud. The new generation of all-purpose crowdsourcing computation platforms such as iExec [8], Sonm [9], and Golem [10] are mainly inspired by Ethereum [24] and convenience of cryptotokens as an economic incentive. We see the common deficiency of these three projects in their approach to the market design. At present, iExec and Sonm offer marketplaces where resources can be manually (or programmatically) chosen. However, we believe that to enable large-scale operation, an automated matching system guaranteeing certain economic performance and fairness would be highly beneficial. ClassAd matching language that iExec uses has similarity with our approach. However, it lacks the ability to prioritize requested resources by weights.

Single-sided cloud auction models in [32], [47] have desirable properties being truthful, combinatorial, and online. However, their design is oriented towards centralized auction, thus being inapplicable to our scenario. The double auction described in [48] is not incentive compatible while [49] relies on broker architecture. [50] concentrates on negotiations scheme for elastic demand, leaving the challenges which DeCloud particularly addresses out of scope. Our idea of clustering bids is close to auction design presented in [51], [52], while due to differences in underlying goods the mechanism design we present is substantially different.

Recently, there were few cases of auction mechanisms designed specifically for the purposes of edge computing. Tasiopoulos et al. [53] build the market taking into account the cellular structure of an underlying network, using Vickrey-English-Dutch auction as a foundation. In their work, the market is split into cells, and latency is the decisive factor. In contrast, DeCloud offers more flexibility, since clients may emphasize any parameters, not only latency, and allocation is performed from the entire scope of the market, not just particular cell. Our approach allows a higher variety of applications to utilize DeCloud, including those initially intended for execution in a cloud environment. Jiao et al. [54] build an edge auction with a particular purpose: offloading PoW mining tasks by mobile blockchain clients. Because of narrow specialization [54] is not generally applicable. Kiani et al. [55] suggest a hierarchical approach to edge resources and devise a revenue-maximizing auction. Contrary to DeCloud, their mechanism is not DSIC, thus potentially being subject to price manipulation attempts.

VIII. Conclusion

Our key contribution is the design of DeCloud, a secure, decentralized and truthful auctioning mechanism that is an essential building block of upcoming open edge computing infrastructures. Our solution is inspired by a distributed trust model that smart contracts can offer, enabling to create a secure sealed bids auction without having a trusted auctioneer as a specific entity. In terms of economic performance, DeCloud achieves between 85% and 75% of its non-truthful benchmark depending on the exact market conditions and losing only between 5% and 0.5% of trades due to the reduction mechanism. In our future work, we plan to run DeCloud on the Ethereum Plasma sidechain or utilize TrueBit's scalable smart contracts.

Acknowledgments

This work was supported by the Academy of Finland in the BCDC (314167), AIDA (317086), and WMD (313477) projects. We would like to thank Rauli Svento for the fruitful discussions and his comments.

[5]The main idea of TrueBit is that instead of collective verification there are *challengers* in the system which verify computation performed by other peers selectively.

REFERENCES

[1] M. Satyanarayanan et al., "The case for vm-based cloudlets in mobile computing," IEEE pervasive Computing, 2009.

[2] F. Bonomi et al., "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.

[3] 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," SIGCOMM Comput. Commun. Rev., 2015.

[4] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in Cloudification of the Internet of Things (CIoT), 2016.

[5] A. Zavodovski, N. Mohan, S. Bayhan, W. Wong, and J. Kangasharju, "Icon: Intelligent container overlays," in Proceedings of the 17th ACM Workshop on Hot Topics in Networks. ACM, 2018, pp. 15–21.

[6] D. Haja, M. Szabo, M. Szalay, A. Nagy, A. Kern, L. Toka, and B. Sonkoly, "How to Orchestrate a Distributed OpenStack," in IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2018, pp. 293–298.

[7] R.-A. Cherrueau, A. Lèbre, D. Pertin, F. Wuhib, and J. Soares, "Edge computing resource management system: a critical building block! initiating the debate via openstack," in The USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18), 2018.

[8] iExec. (2018). [Online]. Available: https://iex.ec/

[9] Sonm. (2018). [Online]. Available: https://sonm.com/

[10] Golem Worldwide Supercomputer. (2018). [Online]. Available: https://golem.network/

[11] N. Szabo. (1996) Smart Contracts: Building Blocks for Digital Markets.

[12] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, Algorithmic game theory. Cambridge University Press, 2007.

[13] T. Roughgarden, Twenty lectures on algorithmic game theory. Cambridge University Press, 2016.

[14] S. Raval, Decentralized Applications: Harnessing Bitcoin's Blockchain Technology. " O'Reilly Media, Inc.", 2016.

[15] Ripple. (2018). [Online]. Available: https://ripple.com/

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[17] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, 2014. [Online]. Available: http://gavwood.com/paper.pdf

[18] R. P. McAfee, "A Dominant Strategy Double Auction," Journal of Economic Theory, vol. 56, no. 2, pp. 434–450, 1992.

[19] D. P. Anderson, "Boinc: a system for public-resource computing and storage," in Fifth IEEE/ACM International Workshop on Grid Computing, Nov 2004, pp. 4–10.

[20] The Docker Project, 2018. [Online]. Available: https://www.docker.com

[21] Intel Software Guard eXtensions, 2017. [Online]. Available: https://software.intel.com/en-us/sgx

[22] ARM TrustZone, 2017. [Online]. Available: https://www.arm.com/products/security-on-arm/trustzone

[23] S. Arnautov et al., "SCONE: Secure linux containers with Intel SGX," in 12th USENIX Symp. Operating Systems Design and Implementation, 2016.

[24] Ethereum. (2018). [Online]. Available: https://www.ethereum.org/

[25] EOS. (2018). [Online]. Available: https://eos.io/

[26] Oraclize. (2018). [Online]. Available: http://www.oraclize.it/

[27] Polkadot. (2018). [Online]. Available: https://polkadot.network/

[28] P. Dütting, T. Roughgarden, and I. Talgam-Cohen, "Modularity and greed in double auctions," in Proceedings of the Fifteenth ACM Conference on Economics and Computation, ser. EC '14. New York, NY, USA: ACM, 2014, pp. 241–258.

[29] R. Panigrahy et al., "Heuristics for vector bin packing," Microsoft Research, 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/

[30] E. Segal-Halevi et al., "SBBA: A Strongly-Budget-Balanced Double-Auction Mechanism," in International Symposium on Algorithmic Game Theory. Springer, 2016, pp. 260–272.

[31] D. Lehmann et al., "Truth revelation in approximately efficient combinatorial auctions," J. ACM, vol. 49, no. 5, pp. 577–602, Sep. 2002.

[32] H. Zhang et al., "A framework for truthful online auctions in cloud computing with heterogeneous user demands," IEEE Transactions on Computers, vol. 65, no. 3, pp. 805–818, March 2016.

[33] Google Cluster Data. (2018). [Online]. Available: https://github.com/google/cluster-data

[34] Amazon EC2 instance types. (2018). [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[35] J. Poon, V. Buterin. (2018) Plasma. [Online]. Available: http://plasma.io/

[36] TrueBit. (2018). [Online]. Available: https://truebit.io/

[37] S. Hu, L. Hou, G. Chen, J. Weng, and J. Li, "Reputation-based distributed knowledge sharing system in blockchain," pp. 476–481, 2018.

[38] R. Dennis and G. Owen, "Rep on the block: A next generation reputation system based on the blockchain," in 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST). IEEE, 2015, pp. 131–138.

[39] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in Proceedings of the 12th international conference on World Wide Web. ACM, 2003, pp. 640–651.

[40] E.-O. Blass and F. Kerschbaum. (2018) Strain: A secure auction for blockchains. [Online]. Available: https://eprint.iacr.org/2017/1044.pdf

[41] X. Li, X. Wang, and F. Liu, "ZebraLancer: Private and Anonymous Crowdsourcing System atop Open Blockchain," in ICDCS, 2018.

[42] CryptoNote. (2018). [Online]. Available: https://cryptonote.org/

[43] Z-Cash. (2018). [Online]. Available: https://z.cash/

[44] Peercoin. (2018). [Online]. Available: https://https://peercoin.net/

[45] Casper. (2018). [Online]. Available: https://github.com/ethereum/casper

[46] Hyperledger Sawtooth Documentation, 2017. [Online]. Available: https://intelledger.github.io/introduction.html

[47] W. Shi et al., "An online auction framework for dynamic resource provisioning in cloud computing," SIGMETRICS Perform. Eval. Rev., vol. 42, no. 1, pp. 71–83, Jun. 2014.

[48] I. Fujiwara, et al., "Applying double-sided combinational auctions to resource allocation in cloud computing," in SAINT, 2010.

[49] P. Samimi et al., "A combinatorial double auction resource allocation model in cloud computing," Information Sciences, vol. 357, pp. 201–216, 2016.

[50] Y. Yang et al., "Double auction and negotiation for dynamic resource allocation with elastic demands," IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, vol. 2015-Decem, pp. 1295–1299, 2015.

[51] X. Feng et al., "TAHES: A truthful double auction mechanism for heterogeneous spectrums," IEEE Transactions on Wireless Communications, vol. 11, no. 11, pp. 4038–4047, 2012.

[52] X. Zhou et al., "TRUST: A General Framework for Truthful Double Spectrum Auctions," in IEEE INFOCOM 2009, April 2009, pp. 999–1007.

[53] A. G. Tasiopoulos, O. Ascigil, I. Psaras, and G. Pavlou, "Edge-map: Auction markets for edge resource provisioning," in 2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM). IEEE, 2018, pp. 14–22.

[54] Y. Jiao, P. Wang, D. Niyato, and Z. Xiong, "Social welfare maximization auction in edge computing resource allocation for mobile blockchain," in 2018 IEEE International Conference on Communications (ICC), May 2018, pp. 1–6.

[55] A. Kiani and N. Ansari, "Towards hierarchical mobile edge computing: An auction-based profit maximization approach," IEEE Internet of Things Journal, vol. PP, no. 99, pp. 1–1, 2017.