FLOCK: Fast, Lightweight, and Scalable Allocation for Decentralized Services on Blockchain

Navin V. Keizer, Onur Ascigil, Ioannis Psaras, George Pavlou
University College London
Email: {navin.keizer.15, o.ascigil, i.psaras, george.pavlou}@ucl.ac.uk

***Abstract*—Many decentralized services have recently emerged on top of blockchain, offering benefits like privacy, and allowing any node in the network to share its resources. In order to be a competitive alternative to their central counterparts, their performance needs to match up. Specifically, service allocation remains a performance bottleneck for many decentralized services.**

In this paper we present FLOCK, an allocation system which is highly scalable, fast, and lightweight. Furthermore, it allows nodes to indicate their preference for clients/sellers without needing to submit bids by using stable matching algorithms. We decouple the price discovery and outsource this function to a smart contract on the blockchain.

Additionally, another smart contract is used to orchestrate the allocation and take care of service discovery, while trusted execution environments securely compute allocation solutions, and off-chain payment networks are used to send rewards.

Evaluation of FLOCK shows that gas costs are manageable and improve upon other solutions which leverage auctions, and that our instance of the stable matching algorithm greatly improves run-time and throughput over auction counterparts. Finally, our discussion outlines practical improvements to further increase performance.

***Index Terms*—Service Allocation, Blockchain, Stable Matching**

I. Introduction

Ongoing efforts aim to decentralize traditionally centralized cloud services, as these introduce a single point of failure and raise privacy concerns. While edge computing [1] has allowed computation to move from the distant cloud closer towards the service user, recent initiatives take it a step further and allow any node in a peer-to-peer (P2P) network to provide a service through its spare resource capacity.

Examples of decentralized services are data storage (Filecoin [2], Storj [3]), computation outsourcing (Golem [4]), and bandwidth sharing (Mysterium [5], Orchid [6]). Empowered by blockchain, these services securely transfer funds as reward for service delivery, without a trusted third party.

An important challenge for the functioning of these services is allocating nodes requiring services to ones that offer the corresponding services, while maintaining decentralization. Although a brute force allocation approach, whereby nodes requiring and providing services individually discover each other, can be viable with limited participants, a *scalable* allocation mechanism is desirable in order for the decentralized services to compete against their centralized counterparts. Decentralization is necessary to protect against bias and censorship from central parties, and protect user privacy.

In conjunction with scalability, two other desirable features for an allocation mechanism are *price derivation* and support for *preference-based* assignment. The price derivation determines appropriate rewards for nodes providing services based on market dynamics. Enabling nodes requesting services to indicate their preference over others in the allocation process is also highly desirable in a decentralized system where any (potentially malicious) node can offer or request services.

These three properties, namely scalability, price derivation, and node preference, are in practice difficult to achieve simultaneously. For example, decentralized auctions, which have been proposed as a solution to this problem, inherently lack sufficient scalability.

In this paper we present FLOCK, an allocation system for decentralized services on blockchain, which captures all three desired properties. An ideal use-case for our system are services with large volumes and tight time constraints such as content retrieval in decentralized storage markets.

Unlike heavy-weight auctions that combine price derivation with node assignment, FLOCK achieves lightweight and scalable allocation by decoupling its price derivation from the preference-based assignment, outsourcing the market function to an oracle smart contract. This oracle sets a global price per service, and can be triggered by any node in the network.

FLOCK uses stable matching algorithms to allow for node preferences, where nodes submit a partial preference ordering of nodes offering or requesting a service to a billboard contract. The computation of matching is outsourced to an off-chain trusted execution environment (TEE). This keeps smart contract costs low, and ensures privacy and speed. The TEE is compensated by the allocated nodes using an off-chain payment network, ensuring speed, low cost, and scalability.

Implementation and evaluation of our contracts shows that gas costs are sufficiently low, especially compared to auction-based solutions. Comparing our proposed algorithm to auction algorithms demonstrates that FLOCK scales much better in terms of computation run-time and throughput.

The rest of the paper is structured as follows. In section II we describe work related to ours, after which we discuss preliminary concepts on which FLOCK relies in section III. Section IV states our system goals, allocation types, and use-cases, before we state our architecture design in section V. We implement and evaluate FLOCK in section VI and discuss possible extensions in section VII. We finally conclude our paper in section VIII.

II. Related Work

In this section we review work related to our system. Stable matching algorithms (section III-D) have been widely studied and applied in real world settings [7]. Examples of this are

978-0-7381-1420-0/21/$31.00 ©2021 IEEE

in cloud resource allocation, student roommate allocation, and hospital resident allocation.

The rest of the related work is divided in two sections: outsourcing computation using TEEs, and auctions on blockchain.

A. Computation Outsourcing Using TEEs

TEEs (section III-C) are used as secure computation enclaves, and are used in a variety of security use-cases, including off-chain computation outsourcing.

One of main downsides of smart contract computation is the lack of confidentiality and privacy, which are essential for use-cases like sealed-bid auctions. Therefore most works optimize privacy rather than scalability. The overheads in these solutions result in slow execution times, and are therefore not suited for fast and scalable allocation.

ShadowEth [8] presents a framework for leveraging TEEs to confidentially execute smart contracts off-chain using a bounty contract on-chain and distributed storage. Airtnt [9] and SPOC [10] outsource computation using TEE's, off-chain payment channels, and a smart contract, targeting fair exchange of resources, security, and correctness. One-to-one allocation of clients to workers is assumed (rather than our multi-input, single output). Ekiden [11] is another system that uses TEEs to address the lack of confidentiality and poor performance of blockchains, by separating execution and consensus.

Finally, Kosto [12] is a framework for secure computation outsourcing using TEEs, and uses a wrapper function for accounting, producing a proof of computation. A broker node allows for minimal open payment channels.

B. Auctions on Blockchain

There has been much work on auction mechanisms. These have traditionally been applied in a centralized manner, as decentralized auctions are difficult to orchestrate. Blockchain has made it possible to implement decentralized auctions on top of smart contracts, either on-chain or outsourced to a dedicated node to keep gas costs low.

Because sealed bid auctions require strong privacy guarantees, they often require expensive cryptographic overheads, lowering their scalability. The following works focus mainly on ensuring privacy and accountability.

AStERISK [13] presents a single-item Vickrey [14] auction on smart contracts. Distributed authority is used for issuing bidding credentials, as well as a number of cryptographic operations. Enkhtaivan et al [15] implement an anonymous English auction using TEE and blockchain, and use group signatures to provide bidder anonymity.

Desai et al. [16] propose a hybrid auction mechanism on blockchain, combining public and private chains, and using simple cryptographic proofs. An auctioneer starts, orchestrates, and deploys the auction. Private chains lower the cost and latency, but the cost of the public portion of the smart contracts and role of auctioneer lower security and scalability.

Galal and Youssef propose a number of sealed bid auctions [17], [18]. Most recently, Trustee [19] presents a Vickrey auction on Ethereum using TEEs. The smart contract is used as a billboard, after which computation is transferred to an enclave using a relay controlled by an auctioneer. The auctioneer can be untrusted and gains no bid information.

The above-mentioned related work on auctions are not applicable for fast and scalable allocation, as they remain expensive in terms of costs and overhead. Besides, they are all single-item auctions, while we need to allocate large numbers of nodes at once. PASTRAMI [20] is a decentralized multi-item auction relying on the Vickrey-Dutch Multi-Item Auction [21] to derive the Vickrey-Clarke-Groves (VCG) equilibrium, allowing multiple buyers and sellers to be matched in one round. After assembling bids on the smart contract, a dedicated node performs the computation to minimize gas fees. All nodes can check the solution and submit proofs of misbehavior. PASTRAMI is still not scalable enough for our use-case as it suffers from high gas fees and latency, as well as from general inefficiencies associated with auctions such as the delay of valuing items and gathering bids.

III. Preliminaries

We now review key concepts on which our solution relies: blockchain, payment networks, TEEs, and stable matching.

A. Blockchain and Smart Contracts

A blockchain, as introduced by Bitcoin [22], is a shared public ledger which is maintained by a P2P network. All nodes have a full view of the shared history, and a consensus algorithm dictates how blocks are appended.

Smart contracts are an extension from transactions, which allow complex scripts of code to be stored and executed on-chain. In Ethereum [23], Solidity, a Turing complete programming language, can be used to create contracts to be deployed to the Ethereum Virtual Machine. After a contract is deployed under its own address, its functions can be called by nodes in exchange for a reward proportional to the load added on the network known as the gas cost. Gas can be bought using the Ether cryptocurrency, and its amount per function is derived from time constraints given by the caller.

Storing data on smart contracts will incur higher gas costs, making alternative decentralized storage solutions such as IPFS [24] more attractive. Accessing read-only data on a smart contract is free. In terms of scalability and usability, keeping gas costs low is essential.

B. Payment Networks

Blockchain applications require low overhead and cost to be useable, and therefore their transaction costs need to remain low. A bottleneck for this has been on-chain payments, which incur a fee for every transaction. When we require many micro-transactions (such is the case for a long-term service) to be sent, the cost quickly becomes unmanageable.

Payment channels have emerged as a useful method to keep the cost low between two parties sending multiple transactions between them. Both parties submit a deposit, send payment receipts, and terminate the channel once finished. However, the cost of opening and closing channels still remains large, and needs to be opened for every new node.

A better extension yet are payment networks [25]–[28]. An example of this is the Raiden Network [1] for Ethereum. These rely on the payment channels, but extend them with cross channel payments, allowing for transactions between nodes who do not have a direct channel. The locked transaction is routed through a path of nodes who have payment channels with enough capacity to support the payment amount. The receiver requests the secret to unlock the payment, which it sends to the closest node (last hop) to receive a balance proof of the payment. This happens between all other nodes on route to finalize the payment. When nodes become unresponsive, the payment can be settled on-chain.

C. Trusted Execution Environments

TEEs allow secure computation to be performed on remote untrusted nodes, using hardware enclaves [29]. The code to be executed in the enclave can be verified for correctness, and external access to the enclave is protected against. Among others (e.g. [30], [31]), Intel software guard extension (SGX) [32] is a TEE implementation which allows users to upload and execute code into a tamper proof secure container, called the enclave. After being uploaded, the code cannot access the OS functions. SGX allows code to be attested to prove it is running properly. This can be done via remote attestation or using intra-attestation. Furthermore, enclave data can be sealed outside of the secure memory.

Although there are many benefits of using SGX, there are known drawbacks as well [33]. First, there is limited secure memory of 128MB. There can also be availability failures as the platform owner can (maliciously) terminate an enclave. SGX can also be susceptible to side-channel attacks [34] and single point attacks. In section V-D we discuss why these drawbacks are not a problem in FLOCK. We use SGX as our TEE in the rest of the paper.

D. Stable Matching Problem

Stable matching algorithms have been used in a wide range of applications [7], as a mechanism to pair entities from two sets based on their preferences for each other, without monetary bids. In its most basic form the problem is also known as the stable marriage problem, with a set of men and a set of women with preference ordering of each man or woman in the other set. Gale and Shapely [35] provided a simple algorithm to produce a stable matching, and the problem has since had many extensions, including partial preference lists, student-roommate allocation, and student-project allocation.

More notably, stable matching has played an important role in hospital resident allocation [36], where the simple problem is extended. Residents are allowed to include in their preference list any subset of the hospitals, and hospitals rank all residents that ranked them. Furthermore, all hospitals have a budget of residents which they can accept.

One feature of stable matching algorithms is that the set of nodes which takes the initiative in the algorithm produces a matching which is best for that set, and thus has an advantage. This yields that the second set is not incentivized to be truthful.

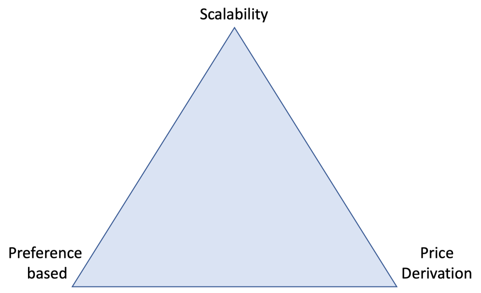

Fig. 1: The allocation triangle, showing the desirable properties for decentralized blockchain-based service allocation.

IV. System Goals

In this section we describe desirable properties our allocation system should have, and discuss how other initiatives have failed to capture these. We then sketch the allocation landscape for decentralized services and derive two types of allocation. Finally, we discuss decentralized storage network use-cases.

A. Desirable Properties

Our allocation should have several properties on top of decentralization. First, our solution needs to be scalable, especially when user numbers grow into the millions (e.g. BitTorrent[2]). This scalability is in terms of latency, throughput, and cost. Second, any node offering a service is potentially malicious. To mitigate against the risk of attacks and insufficient service, nodes need to be able to indicate a preference over their allocated node. Last, as there is no centralized sale of the service, we need some way to set prices. This price derivation should be based on market dynamics to reflect supply and demand, but can be extended to include other factors. Surveying previous works shows that it is particularly difficult to achieve all of these three properties simultaneously. To illustrate we use the allocation triangle in Figure 1.

Normally, decentralized storage networks use an on-chain orderbook to match clients and sellers based on their bid price. This approach would only capture the price derivation aspect of the triangle as it is not scalable and there is no method for specifying a preferred node in a bid. Derived prices are different for individual storage instances (i.e. storage is seen as heterogeneous and can be priced different for different nodes).

Auctions generally improve on this as they capture both price derivation and user preferences through bids. The downside however is their scalability, especially in low latency settings. To start with, the preparation phase of auctions (valuating items and gathering bids) in itself incurs a time delay too significant for instant allocation. The execution phase increases this delay further.

In Vickrey auctions, large numbers of allocations require many subsequent auctions, while the performance of multi-item auctions such as the VCG quickly degrades with increasing numbers of participants and items. These auctions

[1]https://raiden.network

[2]https://www.bittorrent.com/company/about-us/

generally require extra privacy measures to keep bids sealed, as they need to mitigate against a number of attacks. These attacks become less lucrative when no money is involved in the allocation, as is the case with stable matching algorithms. These capture the user preferences, and are faster and more scalable than auctions. In section V-A we combine this with price derivation to achieve all three desirable properties.

B. Types of Allocation

In an allocation, we assume to have a large number of clients and sellers. Generally, the number of sellers will be less than the number of clients. These sellers can offer many different types of goods, depending on the decentralized application. We envision that for each large decentralized application there will be a separate allocation to which clients and sellers flock.

We consider two general service types which need fast, scalable, and lightweight service allocation:

1) Services where a seller sells a single instance of a service. We can denote this service as binary: either it is provided or it is not.
2) Services where the seller sells a part of a good within a certain capacity. Sellers have a budget and try to maximize their revenue by selling in smaller chunks.

These two service types require different modifications to the allocation algorithm used. The first type can be modeled as a regular stable marriage problem with incomplete preference lists, ensuring that nodes do not need to submit preference orderings over large sets of users. This will produce a matching of clients to sellers for the single service instance.

We can model the second type as the hospital/residents problem. This again allows for incomplete preference orderings from the client side, and allows the sellers to define a budget they have available. Clients can be assigned to a seller as long as it has remaining capacity.

C. Decentralized Storage Use-Cases

The service types defined above can be easily illustrated using decentralized storage and retrieval markets. Typically, such networks use the storage market to sell storage to client nodes in chunks (in GB) within their capacity. This is a clear example of our second service type, and can be modeled using hospital/residents allocation.

The retrieval market on the other hand is used by nodes in the network to fetch specific content from one of the storage nodes. In this case the client contracts a retrieval miner, which either delivers the file or not (for which it receives some off-chain payment). This is an example of the first service type.

The rest of the paper focuses on these two service types, and for our evaluation specifically on the second type. We have done this for simplicity and clarity, but our solution can easily be adapted to support more complex cases. One such extension is as follows.

The retrieval market has stringent time constraints, as a client does not want to wait long before fetching a website for example, and hence performing the allocation as requests come in might be too slow. In section VII-C we discuss a method for performing the allocation before requests come in, by allocating retrieval nodes to content providers rather than clients to sellers.

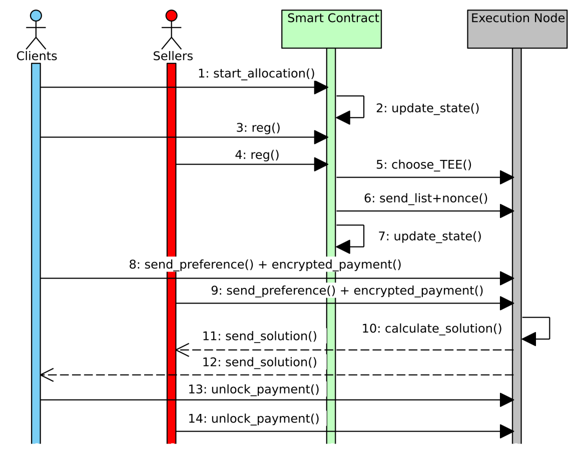

Fig. 2: Simplified sequence of events in our system. We have represented all sellers and clients in single actors.

V. Architecture Design

We now present FLOCK, which satisfies all desirable properties of the allocation triangle of section IV-A for decentralized allocation of blockchain services. We first present an overview of our solution, after which we go into detail of our system components.

A. Overview

Our solution is composed of two parts, which together achieve scalability, price derivation, and preference-based allocation. In the first part, we use a smart contract to function as a billboard for the allocation, which registers participants and orchestrates the initial phase. Service discovery is taken care of by this contract, as it is public and reachable by all nodes. The complexity of this contract is kept to a minimum to save gas costs, and execution of the allocation algorithm is outsourced to an SGX enclave (execution node), which assures privacy and correctness of execution without expensive techniques such as SMPC or zero-knowledge proofs.

To incentivize proper behavior of the execution node, a small reward is transferred by nodes in the allocation using off-chain payment networks. This keeps the cost and overhead low, ensuring scalability.

The algorithms used to compute the allocation are instances of stable matching. Partial preference orderings are submitted based on metrics like personal experience, expected Quality-of-Service (QoS), and reputation. This reputation could be inferred from previous on-chain transactions (like in [37]), or using off-chain reputation systems.

So far, our solution is scalable and allows for preference-based allocation, but lacks price derivation. To solve this, the second part of our solution uses an oracle contract, which decouples the price derivation from the allocation computation. This way, we outsource market function based on macro parameters to the smart contract, rather than do this on a per-item basis, making the process more efficient.

Figure 2 shows a sequence of events during allocation. Any node in the network can trigger an allocation, after which interested nodes register either as a client or seller. After a threshold of nodes is reached, the billboard contract picks an execution node and sends them a list of nodes in the allocation among other parameters. All clients and sellers compile a partial preference list of the nodes in the opposite set, and submit this to the execution node along with an encrypted payment promise. The execution node then computes the allocation and returns the solution to the clients and sellers, who in turn unlock their payments.

Algorithm 1 Setup ($threshold$)

```
Require: $!\ in\_progress$
1: Delete: $clients$, $sellers$, $execution\_node$
2: $THRESHOLD \leftarrow threshold$
3: $c\_count, s\_count \leftarrow 0$
4: $in\_progress \leftarrow true$
5: $waiting\_for\_node \leftarrow false$
```

Algorithm 2 Register (x, $caller\_address$)

```
Require: $in\_progress\ \&\ !\ waiting\_for\_node$
Require: $!\ already\_registered$
1: if $x == 0\ \&\ !\ c\_full$ then
2:     $c \leftarrow c + 1$
3:     $clients.add(caller\_address)$
4: else if $x == 1\ \&\ !\ s\_full$ then
5:     $s \leftarrow s + 1$
6:     $sellers.add(caller\_address)$
7: end if
8: if $c\_Full\ \&\ s\_Full$ then
9:     $waiting\_for\_node \leftarrow true$
10: end if
```

Algorithm 3 ClaimTask ($caller\_address$)

```
Require: $in\_progress\ \&\ waiting\_for\_node$
Require: $!\ already\_registered$
1: $execution\_node \leftarrow caller\_address$
2: $in\_progress \leftarrow false$
```

B. Billboard Contract

The billboard contract orchestrates the beginning phase of the allocation. First, nodes who are interested in participating in an allocation register at the contract as a client or seller. The contract sets parameters to dictate when the allocation is full; this could for example be a time interval (in terms of blocks) or a pre-set number of nodes allowed to join.

After registration, the contract needs to compile a list of participant addresses which the execution node uses to accept user preference lists. In its most basic implementation this temporary storage can be handled by the smart contract, which the execution node can read. However, storing large amounts of data on the blockchain becomes very expensive, and therefore in practice an alternative should be used. We provide a further discussion of this in section VII-A.

C. Integration with SGX

After the registration phase, the smart contract chooses an execution node with an available enclave. This can be an execution node claiming a task, or chosen and contacted by the contract based reputation.

The execution node then obtains a nonce, and a list of node addresses in the allocation. The nonce decides which set in the allocation algorithm will be the initiators for that round. This is not known beforehand so all are incentivized to be truthful in their preferences. Clients and sellers send their preference lists directly to the execution node.

It is assumed that nodes are connected to a payment network such as Raiden, and they can send off-chain payments without added cost. The node in the allocation sends the execution node an encrypted payment receipt, which the latter will be able to unlock upon completion.

Next, the enclave runs the algorithm to produce an allocation. This algorithm will be publicly available (e.g. stored on decentralized storage) and should be remotely attested by a subset of nodes to verify correct instantiation. Finally, the enclave sends the results back to the nodes, after which the secret is released and their balance is updated.

D. SGX Security Analysis

The disadvantages associated with SGX are not a problem for FLOCK. As opposed to auctions, there is little to be gained from attacks on the allocation, as there is no money directly involved. We discuss SGX specific attacks and explain why they are unlikely.

Side-channel attacks aim to infer information from inside an enclave, but in our case the costs of an attack is much higher than potential gain, as price derivation is decoupled and therefore there should be indifference over allocated nodes.

Single-point attacks may be launched as a general attack against a service, aiming to exploit the single point of failure of the enclave, which may lead to censorship or Denial-of-Service. This would require considerable resources, and the smart contract has a number of backup nodes if an enclave fails, and could revert to computing the allocation on-chain.

Attacks from the SGX platform operator are incentivized against as a gas fee is paid to claim the task, which is recovered from off-chain payments. If it acts maliciously it will not recover these funds. Similarly, availability failures should be rare as the platform owner wants to keep the enclave running.

Finally, there is limited protected memory on SGX of 128MB, which poses a constraint on the computation. The required memory is mainly for storing addresses and preference lists. Ethereum addresses consist of 42 hexadecimal characters, which take up 21 bytes. Assuming an average preference list size of 5 per node, the enclave needs to store 6 addresses per node. This means that per node we would need to store 126 bytes, allowing us to fit 1,015,873 addresses in our memory limit. This does not take into account storage of the allocation algorithm or nonce, but as we do not expect even 10% of this amount, we should have plenty of space left. Therefore, the memory limit should not be a problem for our computation.

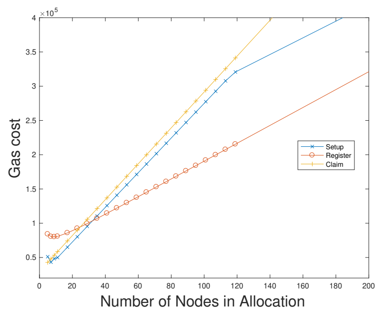

Fig. 3: Progression of gas costs for functions of the billboard contract using on-chain storage of nodes.

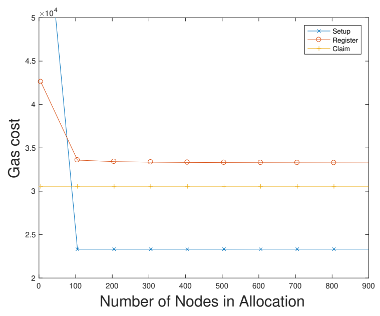

Fig. 4: Progression of gas costs for functions of the billboard contract using off-chain storage of nodes.

	On-chain	PASTRAMI	Trustee
client	143,827	391,046	1,847,178
seller	92,673	95,934	143,720
execution node	-	7,108	-

(a)

Function	Gas Cost	Cost (USD)
deploy contract	213008	\$2.8168
setup oracle	71360	\$0.9437
update oracle	56931	\$0.7529

(b)

Fig. 5: (a) Comparison of individual gas costs of allocations/auctions on blockchain (5 node setup). (b) Gas costs and USD conversions of oracle contract functions.

E. Oracle Contract

The oracle contract is responsible for a global price derivation. The contract derives its state (i.e. the price to be paid for a service) from the underlying billboard contract and other public parameters. The oracle contract can be deployed for different services and incorporate complex financial rules dictating the price. Our simple contract used in section VI-A uses the ratio of clients to sellers and the change of the price of Ethereum to set a price.

The oracle contract updates its price when called by a node. Its pricing mechanism and the parameters from which the price is derived are transparent. Therefore, any node can see if there is a discrepancy between the current price and what it should be, and can call the function to update. This is financially incentivized when the price difference will cause more gain than the cost of calling the contract.

Our work is aimed at fast allocation of large numbers of nodes, and for this use-case global pricing is an efficient solution. However, we note that there is still a need for other mechanisms like auctions for popular items. We envision a secondary market based on multi-item auctions for the top sellers and others who want to sell spare capacity from the primary market. This can be compared to the spot market approach from Amazon, where flat and on-demand instances of services are sold on the primary market based on a global price, after which spare capacity is offered based on bids on the secondary market, usually at a lower price.

VI. Evaluation

To evaluate FLOCK we have implemented[3] its key components and performed extensive simulations to assess latency, cost, scalability and more. Specifically, we look at our smart contract performance in terms of gas costs and extensively compare our proposed allocation algorithm to other initiatives.

A. Smart Contract Cost

We implemented our billboard and oracle contracts in Solidity, and run several simulations to explore their gas costs. We start with our billboard contract, and its main functions: Setup, Register, and Claim. For completeness we have added their pseudo-code. The Setup function (algorithm 1) is used to reset the allocation state and start a new allocation with a new Threshold. Nodes can use Register (algorithm 2) as long as they have not already registered as client or seller, which is checked with an internal function. We keep track of the number of nodes in either set using count variables. Finally, an execution node can use Claim (algorithm 3) if it is not participating in the allocation. Its gas fee can be seen as collateral which it will regain after successful computation.

Figure 3 shows the progression of the gas costs as more nodes participate in the allocation. We clearly see a linear increase of the gas costs, which quickly become unmanageable. This is due to our simple implementation using on-chain storage of the intermediate states (the list of nodes participating in the allocation), and therefore this simple proof-of-concept implementation lacks in performance.

We therefore need an off-chain storage solution in our contract, which we discuss further in section VII-A. We have implemented the billboard contract with this assumption of off-chain storage available in Figure 4, to show that in this case, gas costs are not only an order of magnitude lower, but also remain nearly constant. There will be some interaction needed between the smart contract and the off-chain component, and therefore we can see Figure 4 as a lower bound on gas costs, whilst Figure 3 is the absolute upper bound.

In Figure 5a, we show how our on-chain solution for a 5 node setup outperforms implementations of PASTRAMI and Trustee, for all actors associated with the allocation/auction. For our on-chain implementation, we assume that the gas costs paid by the execution node is fully refunded with an added premium by nodes in the allocation through the payment network, and reflect this in the gas costs. Furthermore, PASTRAMI has an added cost of up to 131,804 in case of misbehavior.

Figure 5b shows the gas costs of the oracle contract, along with the conversion to USD[4]. Deployment and setup of the contract are one-time costs, after which the update cost is paid by those who call it, when they notice a price discrepancy. Nodes are incentivized to call this function when their gain of a new price exceeds \$0.7529. We note that network congestion may increase gas cost significantly, which has prompted layer

[3]https://github.com/navinkeizer/FLOCK

[4]based on the average gas price on 08-11-2020

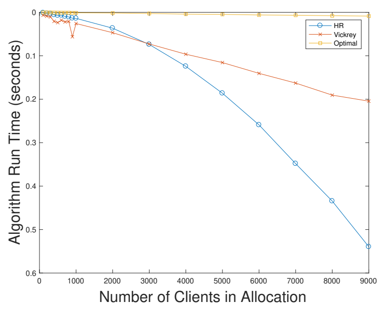

Fig. 6: Single-item allocation algorithm comparison

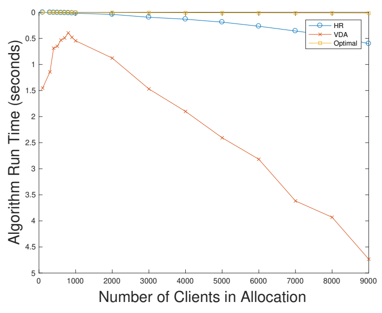

Fig. 7: Multi-item allocation algorithm comparison (maximum capacity = 3)

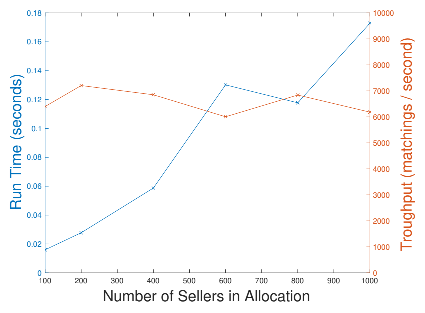

Fig. 8: Performance of Vickrey auction (clients = 500, maximum capacity = 5)

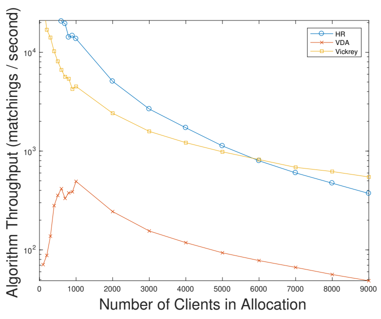

Fig. 9: Multi-item allocation throughput comparison (maximum capacity = 3)

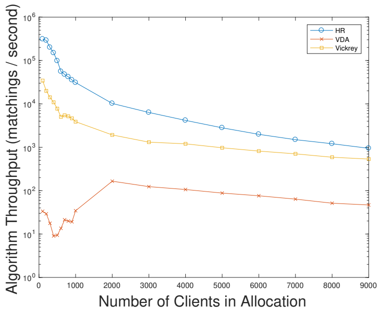

Fig. 10: Multi-item allocation throughput comparison (maximum capacity = 10)

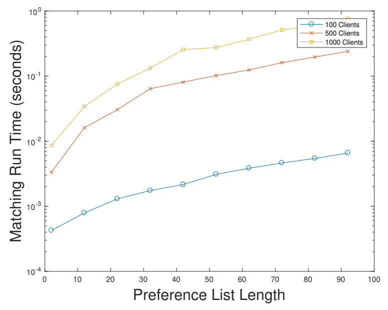

Fig. 11: Influence of preference list size on performance (for hospital/resident algorithm, maximum capacity = 3)

2 [38] scalability solutions to be developed such as roll-ups, shards, and side-chains. On top of Proof-of-Stake based blockchains (like Ethereum 2.0), these are expected to greatly reduce smart contract costs.

B. Allocation Algorithm Performance

To verify the scalability of our system, we show that our allocation algorithm executes with low latency, and achieves a significant improvement over other algorithms that could be used. We focus on the second type of services from section IV-B and model this using an implementation of the hospital/resident (HR) stable matching algorithm[5].

For a comparison with auctions, we use the VDA implementation from PASTRAMI, and implement a simple Vickrey auction, which does not implement the security functions needed for sealed bids. We also looked at implementations of VCG auctions, but as they were significantly slower we have omitted them from our results. Finally, we have implemented a simple mechanism to allocate at random without any constraints, to represent an optimally scalable solution. We have left out the secure enclave overheads as it would be similar for all.

First, we compare the algorithm run times for a constant number of sellers (100) and increasing numbers of clients. We make the assumption that there will be more clients than sellers, as there are barriers to becoming a seller such as spare capacity available and system requirements.

Figure 6 shows the run time for varying clients for the different algorithms in the single-item case – that is, where all sellers only have one item to sell. It is evident that until about 3,000 clients, the HR algorithm performs better than the Vickrey. It is important to note that the strength of the HR algorithm lies in allocating large amounts of nodes at the same time, which is not exploited in the single item case.

Therefore, Figure 7 is more relevant, as it shows the same simulation for the multi-item case (i.e. all sellers can have capacities higher than 1). The HR algorithm clearly outperforms the VDA and remains close to optimal, staying under a second when approaching 10,000 clients.

So far, the HR algorithm is much faster than the VDA algorithm. Looking solely at latency however, gives a skewed image in favor of the Vickrey auction (as we saw in the single-item case). To inspect this comparison further we will now look at the throughput in matchings per second.

Figures 9 and 10 show the throughput of our algorithms for increasing numbers of clients. We have varied the maximum capacity in terms of items to sell per seller, as this illustrates the strength of the HR algorithm. Evidently, the HR algorithm outperforms the VDA algorithm, as well as the simple Vickrey auction up until a certain limit. This limit moves from 6,000 clients for a maximum capacity of 3 to over 10,000 clients for a maximum capacity of 10. This illustrates the scalability of the HR algorithm.

Furthermore, it is important to note that in these simulations the simple Vickrey auction only matches one client per seller rather than within a capacity like the HR, which results in a much lower percentage of clients being matched. A simple solution could be to duplicate the bids for any of the instances

[5]https://github.com/daffidwilde/matching

of a node, but this fails in practice as a client may only require a single instance or would lower their bids for a subsequent instance as its utility for it is lower. Practical implementation would require more sophisticated bids like the VDA, which we have shown performs sub-optimal to the HR algorithm.

Additionally, Figure 8 confirms that if we increase the sellers for the Vickrey auction, to make for a fair comparison with the HR algorithm, its performance lacks behind. We show this by comparing its runtime of 0.12 seconds to match about 150 sellers to 500 clients, to the runtime of the HR in Figure 7, when there are 100 sellers with on average 1.5 items matched to 500 clients, of 0.0066 seconds.

Figure 11 shows the increase of the algorithm runtime as we increase the preference list size and shows a linear increase, confirming that small preference lists are preferred for faster execution, which is in line with users not being able to submit full preference lists due to incomplete information.

Finally, although we cannot show this in simulation, we note that the preparation phases for the auctions are much longer, especially when adding more complex bids. For our HR allocation, we can automate the preparation phase were all nodes have a list of trusted nodes from which it submits those that are available in the allocation. Furthermore, as mentioned before, we compare to a very primitive version of the Vickrey auction which does not implement the security features needed for sealed bid auctions, and therefore its performance can be seen as the ideal case.

VII. Discussion

In our work we have presented a decentralized service allocation system on blockchain, which supports lightweight and scalable pairings of clients and sellers. We have highlighted the importance of capturing all sides of the allocation triangle, in order for decentralized services to compete with centralized counterparts. Our solution captures these aspects, but is by no means the only possible solution. We now discuss possible extensions to our work which could improve on its limitations, and performance.

A. Off-Chain Storage for Intermediate States

Our simple billboard contract implementation is meant as a proof-of-concept. As seen in our evaluation, on-chain storage is not feasible and quickly renders the system useless. Instead, while collecting the nodes in an allocation round, this intermediate data should be outsourced off-chain.

For example, decentralized storage (IPFS, Storj) can be used to temporarily store the user list, as long as retrieval of these nodes remains quick. Another method could be to use another dedicated node with SGX capabilities as a temporary storage node. We leave implementation of this to future work.

B. Decoupled Service Discovery

To further decrease the cost and delays associated with an allocation, we may need a solution to be completely off-chain. We use smart contracts, as their public reachability provides inherent service discovery. This is at a cost, which can be significant if it needs to be repeatedly paid by users.

Therefore, a solution may decouple the service discovery completely, and use other mechanism to query the P2P network and find decentrally orchestrated allocations. Service discovery could also take into account the distance between nodes to minimize networks delays.

C. Real-Time Allocation

Although the performance of our implementation meets our scalability and speed constraints, certain applications require near real-time allocation. The delays including gathering preferences, blockchain consensus, and algorithm execution might be too long. An example of this is content retrieval markets, where nodes requesting some content should not have to wait for the allocation to be ran upon every request.

In this case, we can increase the performance of our system by performing the allocation beforehand, rather than as requests come in. The allocation now is between the content provider and the retrieval miner (seller). This can be done periodically, assigning the task of retrieval of some content to a node. Other nodes can request the content from this node, and payment by clients is still based on the oracle contract. In practice this is an easy extension to our implementation.

D. Oracle Contract Extensions

We have described and implemented a simple oracle contract in this work. This concept can be extended much further. As a start, each decentralized service that is offered may have its own oracle dictating its price. We could take this further and define different prices for different nodes in the same service, based on parameters such as QoS, reputation, and location.

The composition of the contract should be left to developers. Our implementation uses the most basic market dynamics, and lacks financial sophistication. The oracle can be made arbitrarily complex however, and its design may be based on financial incentives and game theory.

Finally, as we increase complexity of our oracle, its cost will increase too. A solution to this could be to outsource its computation to a secure enclave, similarly to how the allocation is computed. This would however decrease the decentralization and security of our solution, and would require more complete security analyses as there might be attack vectors with monetary gain, as in this computation there is an important monetary aspect.

VIII. Conclusion

In this paper we presented FLOCK, a decentralized service allocation system on blockchain, which supports lightweight and scalable pairings of clients to sellers. Our solution leverages smart contracts, stable matching, payment networks, and TEEs to achieve all properties of the allocation triangle.

Evaluation of our system shows that the cost and overhead of FLOCK are manageable and much lower compared to systems using auctions. Furthermore, using a hospital/resident stable matching algorithm greatly improves performance over auction algorithms in terms of latency and throughput. Finally, we discussed additional ways to improve FLOCK's performance and limitations.

References

[1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 01, pp. 30–39, jan 2017.

[2] "Filecoin: A decentralized storage network," Protocol Labs, Tech. Rep., 2017.

[3] "Storj: A decentralized cloud storage network framework," Storj Labs, Inc., Tech. Rep., October 2018. [Online]. Available: https://storj.io/storj.pdf

[4] "Golem whitepaper." The Golem Project, Tech. Rep., 2016. [Online]. Available: https://golem.network/crowdfunding/Golemwhitepaper.pdf

[5] "Mysterium network project whitepaper," Mysterium Network, Tech. Rep., October 2017. [Online]. Available: https://mysterium.network/whitepaper.pdf

[6] J. S. Cannell, J. Sheek, J. Freeman, G. Hazel, J. Rodriguez-Mueller, E. Hou, B. J. Fox, and S. Waterhouse., "Orchid: A decentralized network routing market," Tech. Rep., November 2019. [Online]. Available: https://www.orchid.com/assets/whitepaper/whitepaper.pdf

[7] K. Iwama and S. Miyazaki, "A survey of the stable marriage problem and its variants," ser. ICKS '08. USA: IEEE Computer Society, 2008, p. 131–136. [Online]. Available: https://doi.org/10.1109/ICKS.2008.7

[8] R. Yuan, Y. Xia, H. Chen, B. Zang, and J. Xie, "Shadoweth: Private smart contract on public blockchain," *J. Comput. Sci. Technol.*, vol. 33, no. 3, pp. 542–556, 2018. [Online]. Available: https://doi.org/10.1007/s11390-018-1839-y

[9] M. Al-Bassam, A. Sonnino, M. Król, and I. Psaras, "Airtnt: Fair exchange payment for outsourced secure enclave computations," *CoRR*, vol. abs/1805.06411, 2018. [Online]. Available: http://arxiv.org/abs/1805.06411

[10] M. Król and I. Psaras, "SPOC: secure payments for outsourced computations," *CoRR*, vol. abs/1807.06462, 2018. [Online]. Available: http://arxiv.org/abs/1807.06462

[11] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. M. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *CoRR*, vol. abs/1804.05141, 2018. [Online]. Available: http://arxiv.org/abs/1804.05141

[12] H. Dang, D. L. Tien, and E. Chang, "Fair marketplace for secure outsourced computations," *CoRR*, vol. abs/1808.09682, 2018. [Online]. Available: http://arxiv.org/abs/1808.09682

[13] A. Sonnino, M. Król, A. G. Tasiopoulos, and I. Psaras, "Asterisk: Auction-based shared economy resolution system for blockchain," *CoRR*, vol. abs/1901.07824, 2019. [Online]. Available: http://arxiv.org/abs/1901.07824

[14] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *Journal of Finance*, vol. 16, no. 1, pp. 8–37, 1961. [Online]. Available: https://EconPapers.repec.org/RePEc:bla:jfinan:v:16:y:1961:i:1:p:8-37

[15] B. Enkhtaivan, T. Takenouchi, and K. Sako, "A fair anonymous auction scheme utilizing trusted hardware and blockchain," in *2019 17th International Conference on Privacy, Security and Trust (PST)*, 2019, pp. 1–5.

[16] H. Desai, M. Kantarcioglu, and L. Kagal, "A hybrid blockchain architecture for privacy-enabled and accountable auctions," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 34–43.

[17] H. S. Galal and A. Youssef, "Verifiable sealed-bid auction on the ethereum blockchain," in *IACR Cryptol. ePrint Arch.*, 2018.

[18] H. Galal and A. Youssef, *Succinctly Verifiable Sealed-Bid Auction Smart Contract: ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*, 09 2018, pp. 3–19.

[19] H. S. Galal and A. M. Youssef, "Trustee: Full privacy preserving vickrey auction on top of ethereum," *CoRR*, vol. abs/1905.06280, 2019. [Online]. Available: http://arxiv.org/abs/1905.06280

[20] M. Król, A. Sonnino, A. Tasiopoulos, I. Psaras, and E. Rivière, "Pastrami: Privacy-preserving, auditable, scalable trustworthy auctions for multiple items," 2020.

[21] D. Mishra and D. C. Parkes, "Multi-item vickrey-dutch auctions," *Games and Economic Behavior*, vol. 66, no. 1, pp. 326 – 347, 2009.

[22] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[23] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.

[24] J. Benet, "Ipfs-content addressed, versioned, p2p file system," 2014. [Online]. Available: https://arxiv.org/abs/1407.3561

[25] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," Cryptology ePrint Archive, Report 2019/360, 2019, https://eprint.iacr.org/2019/360.

[26] Y. Sali and A. Zohar, "Optimizing off-chain payment networks in cryptocurrencies," 2020.

[27] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," 05 2019, pp. 106–123.

[28] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," Tech. Rep., January 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[29] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.

[30] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," ser. STC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 21–30. [Online]. Available: https://doi.org/10.1145/1456455.1456460

[31] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Trusted execution environments on mobile devices," in *2013 ACM SIGSAC conference on Computer communications security, 4-8 November 2013, Berlin, Germany*. ACM, 2013, pp. 1497–1498.

[32] "Software guard extensions programming reference, revision 2," Intel, Tech. Rep., 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[33] Z. Bao, Q. Wang, W. Shi, L. Wang, H. Lei, and B. Chen, "When blockchain meets sgx: An overview, challenges, and open issues," *IEEE Access*, vol. 8, pp. 170 404–170 420, 2020.

[34] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Sgx-tor: A secure and practical tor anonymity network with sgx enclaves," *IEEE/ACM Transactions on Networking*, vol. 26, no. 05, pp. 2174–2187, sep 2018.

[35] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[36] A. E. Roth and E. Peranson, "The redesign of the matching market for american physicians: Some engineering aspects of economic design," *American Economic Review*, vol. 89, no. 4, pp. 748–780, September 1999. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/aer.89.4.748

[37] N. V. Keizer, O. Ascigil, I. Psaras, and G. Pavlou, "Rewarding relays for decentralised nat traversal using smart contracts," in *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, ser. Mobihoc '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 309–314. [Online]. Available: https://doi.org/10.1145/3397166.3412799

[38] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Off the chain transactions." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 360, 2019.

[39] K. Karantias, A. Kiayias, and D. Zindros, "Smart contract derivatives," Cryptology ePrint Archive, Report 2020/138, 2020, https://eprint.iacr.org/2020/138.

[40] E.-O. Blass and F. Kerschbaum, *Strain: A Secure Auction for Blockchains: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, 08 2018, pp. 87–110.

[41] D. C. Sánchez, "Raziel: Private and verifiable smart contracts on blockchains," *CoRR*, vol. abs/1807.09484, 2018. [Online]. Available: http://arxiv.org/abs/1807.09484

[42] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," *IBM J. Res. Dev.*, vol. 63, no. 2–3, Mar. 2019. [Online]. Available: https://doi.org/10.1147/JRD.2019.2913621