A Customizable dApp Framework for User Interactions in Decentralized Service Marketplaces

Veno Ivankovic*, Zeshun Shi*, and Zhiming Zhao*
*Informatics Institute, University of Amsterdam, Amsterdam, 1098 XH, the Netherlands
Email: ivanvenoivankovic@gmail.com, z.shi2@uva.nl, z.zhao@uva.nl

***Abstract*—Blockchain technology has been utilized in many business cases due to its capability for the development of trustless systems. There is a huge potential for building service marketplaces on top of blockchain technology as decentralized applications (dApps). In such dApps, the point is to exchange and purchase assets and record these transactions on the blockchain to improve the transparency and trust of the marketplace. This work presents a software framework and describes the software prototype implementation, which allows for the provisioning of services on a dApp. The interactions between providers and customers involved in the procurement of services traded on the marketplace are recorded on a distributed ledger. In our dApp, services are provisioned via a configurable auctioning subsystem. Furthermore, after an auction for a service is finished, a Service Level Agreement (SLA) is finalized between a provider and customer. We include a decentralized witness monitoring subsystem to detect Service Level Objective (SLO) violations on this SLA, and witnesses participating in SLA monitoring earn token rewards for their service. Finally, we demonstrate the feasibility of our prototype using state-of-the-art smart contract testing methods.**

***Index Terms*—Blockchain, Decentralization, Service Marketplace, dApp**

I. Introduction

Decentralized Applications (dApps) are applications that do not interact with traditional centralized computing technologies but rather read data from and write data to a blockchain [1], [2]. Due to the fact that blockchains are managed by peer-to-peer networks, modifications to data on the blockchain are typically considered more trusted and secure than in traditional architectures[3], [4]. Blockchain-based dApps rely on smart contracts, which are computer programs that run on the blockchain and perform actions based on inputs without intermediate processing. The marketplaces for cloud services are of particular interest since cloud services are generally provisioned by large providers on a single platform [5], [6], [7]. Examples of this are Azure, AWS, and Google Cloud, among others[8]. Because of this, when customers are purchasing cloud resources to satisfy application requirements, they are often limited to single vendors, especially if they have already purchased resources from a particular vendor. We address this issue by designing and developing a dApp prototype for a marketplace in which services are provisioned in a fair and transparent way via auctions[9].

When cloud resources are provisioned, they usually have associated with them a Service Level Agreement (SLA) which is a legal contract that specifies the conditions under which that resource is delivered to the customer by the provider [10]. SLAs are typically composed of Service Level Objectives (SLOs), which are specific stipulations in SLAs. Research has been done on implementing SLAs as smart contracts and automating the payout process in case violations are detected. However, these violations are often reported by a single trusted third party (also known as an oracle). This goes against the decentralized promise of dApps due to the single point of failure vulnerability of oracles [11]. As such, the dApp developed for this paper addresses this by allowing multiple oracles to report violations and thus reach a consensus, and earn rewards for this monitoring service.

In this paper, we present an integrated software framework that can be deployed as a dApp for the provisioning and monitoring of services on a decentralized marketplace. The framework is purposefully generic and loosely coupled. This is done so that it can be deployed for research purposes or can be used by dApp developers to instantiate their own specialized dApp marketplaces. This specialized dApp marketplace could be a cloud service marketplace, a data marketplace, a software development marketplace, or any other marketplace where services are provisioned via auction and monitored according to the aforementioned decentralized monitoring mechanism. This framework includes both the smart contract logic run on the blockchain and the software which allows for interaction with the smart contract logic.

A. Problem Statement

In a centralized service marketplace, the interactions between users rely on a trusted third party (TPP) that operates the platform. This causes several problems:

1) All participating parties must trust that the marketplace will be efficient and fair. However, this TPP has its own economic interests and, as such, may be unfair when facilitating transactions between users in order to maximize its own profits [12]. For example, the TPP may operate as a market exchange intermediary and charge users an unnecessary commission when they are buying or selling services. This reduces the margin of profit for market participants.
2) In a centralized marketplace, the TPP may unilaterally enforce arbitrary rules on the platform participants at any time. This may include blocking certain users from participation or changing the rules which dictate how services are bought and sold. Conversely, the rules of

decentralized marketplaces are dictated by immutable smart contracts [13].

3) In centralized marketplaces, participants do not own any of their data. Personal information, reviews, and purchase history are owned by the reputable TPP. This can cause issues; for example, if the user wishes to migrate their operations to another platform, they may not have the right to request the TPP to delete their data. In a decentralized marketplace, each user controls their own data, which facilitates trust in the system and eases privacy concerns [14].

In a decentralized marketplace, the only parties are the buyer and seller. At the same time, the market is simply a set of program instructions that automates the transaction, thereby removing the TPP [15]. Therefore, the focus of our software framework is primarily on the user interactions related to SLA instantiation and SLA monitoring on a dApp service marketplace. This poses the following research question:

- RQ. How to design and implement a flexible software framework for a decentralized service marketplace that facilitates trustworthy business interactions between users?

This main research question serves as an overall guide for this paper, including how the software should be designed to offer support for the interested parties and what the interactions between those parties actually are. This research question, therefore, invites the following sub-questions:

- RQ1. How to design and implement a software prototype for a decentralized service marketplace?
- RQ2. How should we design user interfaces and smart contract logic to provide services between customers and providers on our decentralized marketplace via auctions?
- RQ3. How should our decentralized marketplace and user interfaces be designed and implemented in order to facilitate trustworthy SLA monitoring?
- RQ4. How to evaluate the success of the developed software prototype?

B. Contributions

The main contributions of this paper can be summarized as follows:

1) We present a design and implementation for a fully integrated, loosely coupled software framework for a decentralized service marketplace. This includes both the on-chain smart contract logic and the off-chain external application. This software framework can be used and adapted by dApp developers as a starting point for further research or industrial development.
2) The software framework includes a configurable auctioning subsystem for service provisioning. As such, the framework provides users with an effective and trustworthy mechanism to achieve market exchange.
3) The software framework also includes a decentralized SLA monitoring subsystem. This allows providers and customers to configure the rules under which their SLA is monitored, and allows SLA monitoring witnesses to earn rewards for their monitoring service.

The rest of the paper is organized in the following way. In Section II we present the design of the decentralized service marketplace which we developed. In Section III the implementation details of our solution are explained. In Section IV the experimental research and its results are provided. Section V surveys the related work. Finally, in Section VI we provide our conclusions and future works.

II. Framework Overview

In this section, we will explain the system overview. The section is started with an analysis of the actors. Next, the system architecture is described in detail.

A. Actor Analysis

Actors which interact with the system are human roles, external systems, or devices that exchange information with the dApp. With this in mind, we identify the following actors:

1) Service Customers: Service customers use the dApp to find providers that can offer them services. They should be able to make requests on the platform for services they require and enter an SLA with a service provider. They pay for these services but can receive compensation in case of SLA violations.

2) Service Providers: Service providers use the dApp to list their available services on the platform and bid on customer-initiated service requests. They earn monetary rewards for these services from customers but may be penalized in case of SLA violations.

3) SLA Monitoring Witnesses: Witnesses can use the dApp to monitor an SLA and receive monetary compensation or punishments when reporting violations correctly or incorrectly.

4) dApp Developers and Operators: Developers can use and modify the developed dApp framework for specific business use cases. They can also perform auxiliary operational tasks.

B. System Architecture

We can now define the architecture of the system we are developing. We present our architecture from two viewpoints. The first is an enterprise viewpoint for managers and stakeholders, while the second is a computational viewpoint for developers.

1) Enterprise Architecture: Let us examine the architecture from an enterprise viewpoint. Namely, we want to provide an overview of the system from a managerial perspective. In this way, we show stakeholders, key subsystems, and their interactions to provide a holistic outline of the system. This architecture diagram can be viewed in Figure 1.

The two subsystems stakeholders interact with are the auction subsystem and the decentralized witness SLA monitoring subsystem. Understanding these two dApp subsystems is critical to understanding the dApp functionality and how stakeholders use the dApp, so let us examine them more closely:

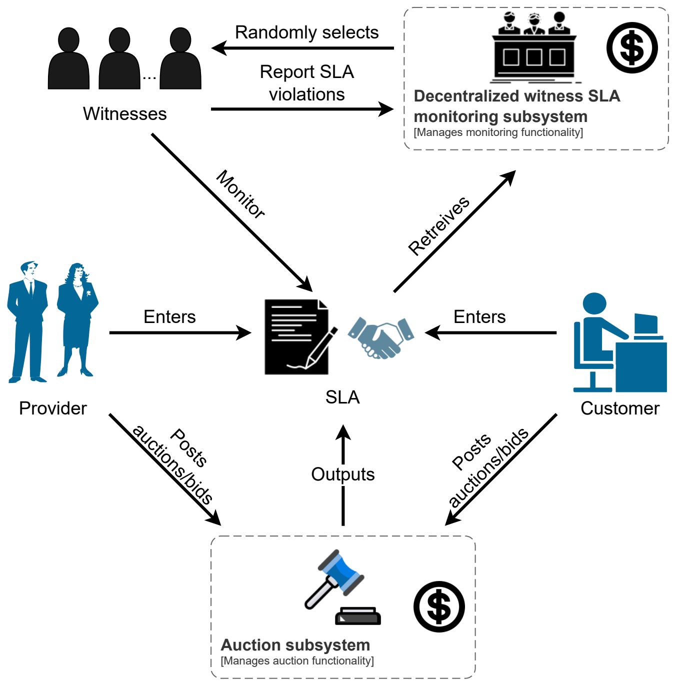


Fig. 1. An enterprise architecture diagram of the developed dApp framework.

- **Auction Subsystem**: This subsystem manages the auction functionalities in the dApp. As such, service providers and customers can use it to post both auctions and bids (as we implement both forward and reverse auctions). The data posted is recorded on the blockchain by invoking the smart contracts on the auctioning subsystem. After this, the auction ends, and a winner is declared according to the rules specified by the auction initializer. Then, the winning bidder and auction initializer enter an SLA. It should be noted that there are hundreds of auction models that exist and can be easily implemented using smart contracts. In this paper, we choose to implement four of the most common auction models (i.e., English, Dutch, first-price sealed-bid, and second-price sealed-bid auctions.) and their reverse variants. This subsystem automates payment between provider and customer for the service after a winner has been declared.
- **Decentralized Witness SLA Monitoring Subsystem**: When an auction finishes and a new SLA is posted onto the dApp, it is retrieved by the decentralized witness SLA monitoring subsystem. It first randomly selects from a pool of witnesses the appropriate number of witnesses to monitor the SLA according to the SLA terms. These witnesses then concurrently monitor the SLA in order to detect SLA violations for the duration of the service [16]. They report SLA violations which are written to the blockchain via smart contracts. Once an SLA finishes, the subsystem automates payment to the appropriate parties. This means that the SLA parties pay witnesses monitoring fees, the provider pays the customer violation fine amounts in case violations were detected, and witnesses pay punishment fees to SLA parties in case they voted on violations in a way that contradicted the voting consensus.

2) Computational Architecture: We can also draft our architecture from a computational and data viewpoint. The goal of this architecture is to visualize how system components are coupled as well as visualize how data flows between components. Our computational architecture diagram can be viewed in Figure 2. Let us examine the design of the system. It can most clearly be decomposed into three modules: the graphical user interface module, the communication module, and the smart contracts module.

- **Graphical User Interface (GUI)**: The GUI consists of three submodules, which are GUIs for providers, customers, and monitoring witnesses. From the diagram, we see that the provider and customer UIs can use the GUI to send and receive auction data, and the witnesses can send and receive monitoring data. Conceptually, this can be grouped into user input and output (IO) data. It should be noted that the user IO data also includes authentication data so that the communication module can correctly send and receive data to and from the appropriate users.
- **Communication Module**: The communication module is primarily responsible for GUI to smart contract communication. Once the communication module receives the user IO data, it first checks the authentication data sent by the user. An identity management submodule

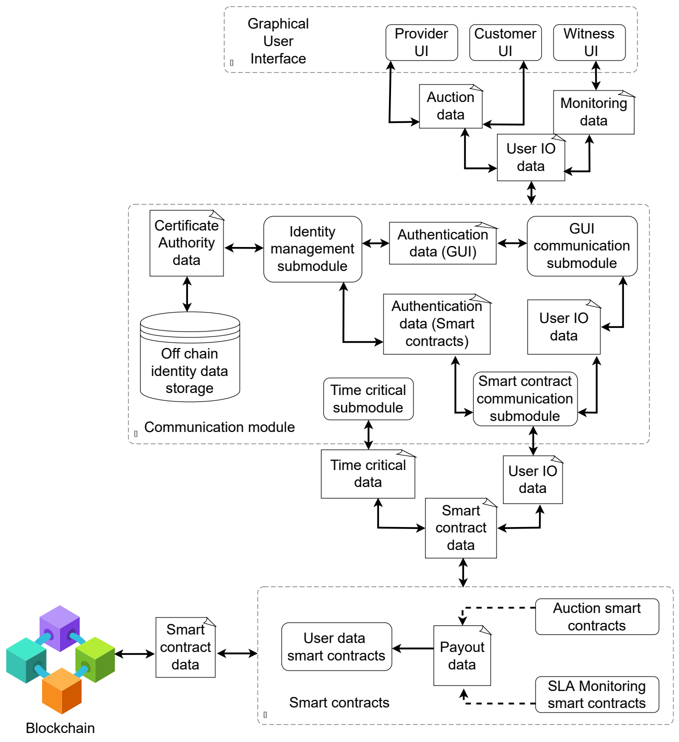


Fig. 2. A computational architecture diagram of the developed dApp framework.

maps the authentication data sent by the user to the off-chain identity data storage, where it checks if the user is registered on the dApp. If this check is successful, then the GUI communication submodule can send user IO data to the GUI, or alternatively, the smart contract communication submodule can send user IO data to the smart contract module. We should also mention the time-critical submodule, which regularly checks the smart contracts to determine if any deadlines have passed (such as an auction or SLA ending).

- **Smart Contracts**: This module contains the smart contract logic of the dApp, which is hosted on the peer-to-peer networked blockchain. The smart contract logic handles the auction data processing and SLA monitoring processing. Furthermore, it handles events like payouts and submits the appropriate funds to the users' account balance data. These account balance changes can be a result of the auction and SLA monitoring smart contracts or sent directly by the user in case of deposits and withdrawals to the users' account balance. As such, the smart contracts module handles data exchange between the communication module and the data stored on the blockchain.

III. Implementation and Demonstration

This section describes how we implemented the dApp prototype based on the design in Section II. The link to our Github repository can be found here[1].

A. Design Choices

When discussing the choice of technologies, it is best to refer to the technology which allows us to implement each module in the design, namely the blockchain, smart contracts, communication module, and GUI. These technologies then contribute to the overall implementation of the architecture according to our design.

1) Blockchain: The technology we have opted for to build the blockchain infrastructure is Hyperledger Fabric (HLF)[2]. It makes use of peer nodes that maintain the ledger and execute smart contract code, orderer nodes that maintain the consistency of the ledger world state, and membership service providers which use cryptographic methods to authenticate users. In terms of technical considerations from the literature study, we found that HLF is the best platform for the development of dApps due to its best performance for success rate, average latency, throughput, and resource consumption KPIs. This is primarily because it is designed for business use cases that require permissioned blockchains. To store the world state of the ledger CouchDB is used. HLF provides us with SDKs for communication with the smart contracts, and in this case, the Node.js SDK is used. Finally, HLF allows us to write smart contracts in Javascript, which is a very common general-purpose programming language. Because of the high performance and large tooling available this technology was chosen.

2) Communication module: The communication module is implemented in Express.js[3], which is a framework for Node.js web applications. Express.js is one of the most popular web development frameworks as it is both minimal and flexible. It serves as the middle point of contact between the blockchain and the GUI and as such, acts as an API server. The API server functions as a RESTful server that follows the HTTP protocol. This technology was primarily chosen because of its scalability (Node.js can handle 15000 requests per second) and ease of implementation. Another motivation for using this technology is that since the smart contracts are written in Javascript, it is also beneficial to write the communication module code in Javascript, as this allows us to use the Hyperledger Fabric Node SDK for communication with the smart contracts. Finally, since the GUI is implemented as a web app JSON data is sent to the communication module, which can be easily parsed by the API server.

3) GUI: The user interface is implemented as a web app, meaning that the languages used to program the web pages are HTML5, CSS, and Javascript. An Express.js server is used to serve the web pages. Implementing the GUI as a web app was chosen as web apps are very accessible and can be used on almost all devices with a web browser. To make API requests on each web page, axios[4] and Vue.js[5] are used, in which axios is used to send and receive HTTP requests and responses, and Vue.js to easily parse and dynamically render the JSON data which was received.

B. Smart Contracts

HLF keeps track of data as assets, meaning that we can keep track of which assets have been posted onto the system and make modifications to those assets. In our case, assets are auctions, SLAs, and user data such as account balance. A lot of our functionalities depend on timed events; however, it is not possible to call certain functions from within the smart contract. Therefore, we instead keep track of timed events as deadline data, and this data is queried periodically by the API server. It is also possible to determine from within a smart contract which user invoked a certain smart contract function. This allows us to return the data specific to the permission that the user should have. For example, when querying an SLA as a witness, users cannot see which other witnesses have already submitted violation votes. The smart contracts were implemented in Javascript. The main smart contract functions are listed in Table I. Next, let us examine the smart contract for the three submodules in Figure 2, namely user data, auctions, and SLAs.

1) User Data: The blockchain stores account balance user data account as a simple integer value representing tokens. There are two ways this account balance can be changed, namely by the owner of the account or from within the

[1]https://github.com/venoivankovic/awesome_repo
[2]https://hyperledger-fabric.readthedocs.io/en/latest/
[3]https://expressjs.com/
[4]https://axios-http.com/docs/intro
[5]https://vuejs.org/

TABLE I
SMART CONTRACT FUNCTIONS OF THE DAPP PROTOTYPE.

Function Name	Function Description
Register User	Registers a user onto the dApp.
Query User Data	When queried, return users' information (e.g., their identity and account balance).
Submit Auction	Submits an auction object onto the ledger.
Submit Bid	Submits a bid object onto the ledger for a certain auction.
Query Auctions	Returns to the user auctions for which they are eligible as a bidder.
Query My Auctions	Returns to the user auctions which they have previously submitted.
End Auction	It can only be invoked with the credentials of the auction initializer or by the API server.
Query SLAs	Return to the user SLAs where they are either a provider, customer, or witness.
Submit Vote	Submits an SLO violation vote onto the dApp.
End SLA	Ends an SLA. This can only be invoked by the API after the SLA duration has passed.
Withdraw Funds	Withdraws funds from a user's account balance.
Add funds	Adds funds to a user's account balance.

smart contract. We allow users to deposit and withdraw funds from their accounts as this is both realistic and helpful for debugging. Importantly, users can only add or withdraw funds from their accounts, not from other accounts. Furthermore, in case of on-chain events such as an auction ending or SLA violations occurring, smart contract functions can access the appropriate user data and add or take away funds to and from their account. This serves to automate payments and simulate a realistic payout scenario, which is useful when developing a marketplace. We have decided to name the tokens used to represent account balance *zCoins*. In the current implementation, this simulated account balance token exists for our experimental purposes. HLF is designed to be a highly modular blockchain platform, so an encrypted cryptocurrency component can be easily added to the original platform.

2) Auctions: The smart contract also allows auctioning functionality. More specifically, customers and providers can initialize auctions on the dApp and allow the bidding party to view them and bid on them. We store the data, not the metadata, on the blockchain. This is primarily due to the permissioned blockchain used and the fact that the data is not too large. Furthermore, it allows us to process the data on the blockchain directly with the functions of the smart contract and as such enhance trustworthiness. When auctions end, the smart contract functions generate an SLA according to the specific auction rules without any user interference, as well as awarding the service fee to the provider immediately.

3) SLAs: Finally, the smart contract supports SLA monitoring. The SLA customer and provider can query the SLA and view its data, as can the witnesses selected for the SLA monitoring, but other users cannot, to maintain privacy. When the auction is ended a smart contract function instantiates the SLA and selects the witnesses to monitor the SLA in an unbiased way [16]. Those witnesses can submit votes to the SLA for which they were selected. Since the SLA monitoring contains many timed event transactions (such as votes and SLA deadlines), timestamps are stored on the blockchain data. This data is then periodically queried by the API server to determine if functions such as ending the SLA should be invoked. When the SLA is ended, the smart contract processes its data and invokes the appropriate payouts to the users' account balance.

C. Express.js API Server

When the API server receives incoming requests, it first checks if the request contains a JSON Web Token (JWT). If no token is given, it checks if the requests contain data that signifies that the user is registered on the dApp. If this is the case, a new JWT is generated for the user and sent back to the web app. This allows users to log in to the web app and browse the dApp. Once the user is authenticated the API server can perform smart contract invocations on their behalf via the cryptographic wallets stored in the API server file system. This allows the communication module to relay user IO data between the smart contracts and the GUI.

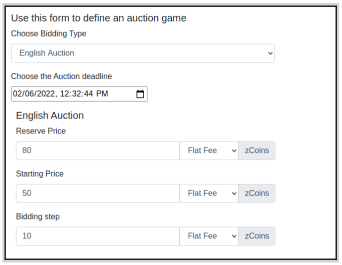
The API server also stores the admin ID and uses this identity to call certain functions on behalf of the dApp, such as checking if an SLA deadline has passed and ending the SLA if this is true, or doing the same for auctions. These types of auxiliary functionalities are done asynchronously to prevent high loads on the server.

D. Web App

The GUI is used by service providers, customers, and witnesses. The GUI point of entry is a login page. If the user has already been registered, they can then login and the API server sends back their authentication token. Each user type User Interface (UI) consists of web pages where users can perform their specific user tasks. Let us examine the GUIs in more detail. We first describe the Provider and Customer GUI and then the Witness GUI.

1) Provider and Customer UI: The provider and customer UIs are relatively similar and symmetrical, given that both user types can act as auction initializers and bidders. As such, we have grouped their UIs when discussing them. Both UIs are web apps and consist of web pages where users can fulfill their requirements. Let us describe each of the pages to demonstrate how they allow users to perform tasks on the dApp.

- **Homepage**. Once users log in to the web app with their identity, their landing point is a homepage. Here users are greeted with a welcome card that gives them important information about how to use the dApp. Furthermore, users can also use this web page to view their user-specific data, namely their account balance. We also include interactivity on this page; namely, from here, users can deposit or withdraw funds from their account balance, leveraging the add and withdraw functions from the smart contract. When a user makes a deposit, the



Use this form to define an auction game

Choose Bidding Type

English Auction

Choose the Auction deadline

02/06/2022, 12:32:44 PM

English Auction

Reserve Price

80	Flat Fee	zCoins

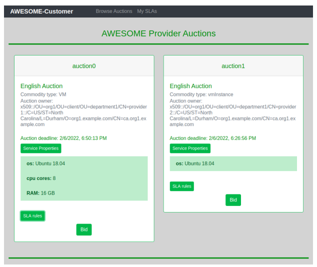
Starting Price

50	Flat Fee	zCoins

Bidding step

10	Flat Fee	zCoins

(a) Post Auction



AWESOME Provider Auctions

auction0

English Auction

Commodity type: VM

Auction owner: x509::/OU=org1/OU=client/OU=department1/CN=provider1::/C=US/ST=North Carolina/L=Durham/O=org1.example.com/CN=ca.org1.example.com

Auction deadline: 2/6/2022, 6:50:13 PM

Service Properties

os: Ubuntu 18.04

cpu cores: 8

RAM: 16 GB

SLA rules

Bid

auction1

English Auction

Commodity type: vmInstance

Auction owner: x509::/OU=org1/OU=client/OU=department1/CN=provider2::/C=US/ST=North Carolina/L=Durham/O=org1.example.com/CN=ca.org1.example.com

Auction deadline: 2/6/2022, 6:26:56 PM

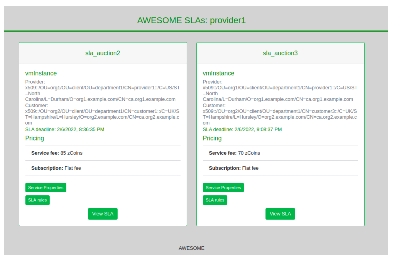
Service Properties

os: Ubuntu 18.04

SLA rules

Bid

(b) Browse Auction



AWESOME SLAs: provider1

sla_auction2

vmInstance

Provider: x509::/OU=org1/OU=client/OU=department1/CN=provider1::/C=US/ST=North Carolina/L=Durham/O=org1.example.com/CN=ca.org1.example.com

Customer: x509::/OU=org2/OU=client/OU=department1/CN=customer1::/C=UK/ST=Hampshire/L=Hursley/O=org2.example.com/CN=ca.org2.example.com

SLA deadline: 2/6/2022, 8:36:35 PM

Pricing

Service fee: 85 zCoins

Subscription: Flat fee

Service Properties

SLA rules

View SLA

sla_auction3

vmInstance

Provider: x509::/OU=org1/OU=client/OU=department1/CN=provider1::/C=US/ST=North Carolina/L=Durham/O=org1.example.com/CN=ca.org1.example.com

Customer: x509::/OU=org2/OU=client/OU=department1/CN=customer3::/C=UK/ST=Hampshire/L=Hursley/O=org2.example.com/CN=ca.org2.example.com

SLA deadline: 2/6/2022, 9:08:37 PM

Pricing

Service fee: 70 zCoins

Subscription: Flat fee

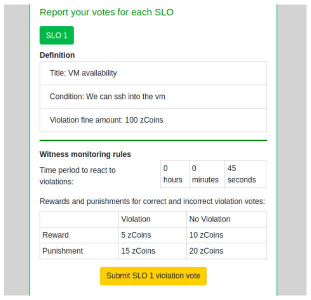
Service Properties

SLA rules

View SLA

AWESOME

(c) My SLAs



Report your votes for each SLO

SLO 1

Definition

Title: VM availability

Condition: We can ssh into the vm

Violation fine amount: 100 zCoins

Witness monitoring rules

Time period to react to violations:	0 hours	0 minutes	45 seconds

Rewards and punishments for correct and incorrect violation votes:

	Violation	No Violation
Reward	5 zCoins	10 zCoins
Punishment	15 zCoins	20 zCoins

Submit SLO 1 violation vote

(d) Monitoring Card

Fig. 3. The web app demonstration of the framework.

deposit amount is then added to their account balance and the result is displayed on the homepage.

- **Post Auction**. The next page we will examine is the post auction page. Here, the user defines the service properties, the SLOs, the monitoring rules, and the auction rules. This information is filled out using common HTML forms, which become JSON data when submitted. The page where auctions are posted is the page with the most user inputs, as it is here that the possible configurations are defined. In Figure 3a, we can see how the auction is configured. There is a button at the bottom of the post auction page which allows users to post auction objects onto the dApp for bidding.
- **Browse Auctions**. According to our design, users of the customer type can view and bid on provider-initiated auctions and vice versa. Essentially, on this page, customers can see a list of provider-initiated forward auctions, and providers can see a list of customer-initiated reverse auctions. As such, after a provider submits the auction a customer can view this auction on their Browse Auctions page, as well as any other auctions which have already been posted, as shown in Figure 3b. For each auction, the customer can see the auction type, the service type, the auction initializer, the auction deadline, the service properties, and the SLOs. Each auction also has a bid button that, when clicked, leads the user to the auction card page for bidding.
- **My Auctions**. Auction initializers can view auctions they posted onto the auction. The layout is very similar to the Browse Auctions page, but they can also see the highest bid currently submitted to the auction. Clicking on the view button leads auction initializers to the auction card page.
- **Auction Card**. Each auction card gives users all the required information about that auction. If they are the auction initializer, they may have the option to end the auction at this point or allow it to end automatically if they added an auction deadline. If they are the bidder, they can use this page to submit a bid for the service being auctioned.
- **My SLAs**. Once an auction ends, either from an auction initializer input or from the auction ending due to a deadline, then providers and customers can view the SLA on their My SLAs page, as shown in Figure 3c. This layout is similar to the *Browse Auctions* and *My Auctions* pages, except that the items displayed are SLAs.

2) Witness UI: To facilitate proper SLA monitoring, we require a UI for witness monitoring.

- **My Monitoring Jobs**. Similar to the provider and customer, witnesses can also log in to view their user data, including the monitoring task and account balance. Furthermore, if they select the “My Monitoring Jobs” tab, they can see active SLAs for which they were randomly selected to monitor for SLA violations. This tab is similar in layout to the My Auctions, Browse Auctions, and My SLAs tabs. Each monitoring job leads to a Monitoring card.
- **Monitoring Card**. Once a monitoring job has been selected, the witnesses can submit SLO violations. In Figure 3d, we can see the monitoring card. This shows an SLO for which a witness can submit violations if they detect any. The dApp does not allow witnesses to report an SLO violation more than once within a time window specified by the auction initializer when defining the SLA monitoring rules.

IV. Evaluation and Validation

In this section, we are concerned with performance and quantitative measurements we can make for the smart contracts to test the scalability of the dApp. Because business transactions such as submitting auctions, bids, and violation reports are submitted via smart contracts onto the blockchain, to demonstrate the feasibility of the smart contracts, it is necessary to benchmark and test their performance. We use Hyperledger Caliper[6], a tool for testing Hyperledger Fabric smart contracts.

A. Experimental Setup

It must be stated that our smart contracts can be subdivided into three operational types, all of which we must test. Namely, some functions are create operations, some functions are read operations, and some functions are update operations. To

[6]https://hyperledger.github.io/caliper/

further clarify this, assets are stored on the blockchain in an append-only structure as well as in a CouchDB database to keep track of the current world state. These CouchDB assets are stored in key-value pairs. Create operations place key-value pairs onto the CouchDB database. Read operations use a key to retrieve the value associated with that key. Update operations use a key to retrieve the value associated with a key, make changes to that value and then write the new key-value pair onto the CouchDB database and blockchain.

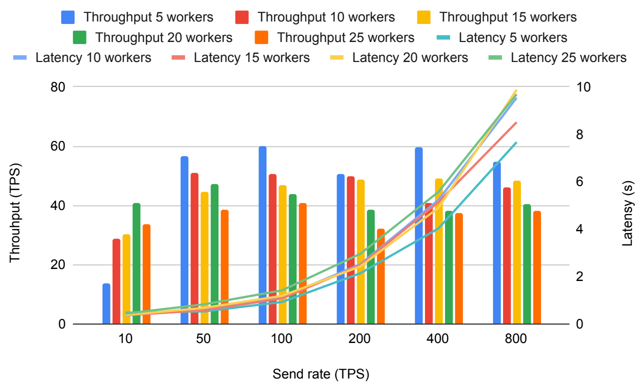
During our experimental runtime, we manipulated the number of workers, meaning the number of concurrent processes performing the Caliper test. We also manipulated the rate at which transactions are submitted. It should be noted that during experimentation, the configured rate control may deviate from the actual rate at which transactions are submitted due to computational factors. The KPIs which we measure are throughput, meaning the rate at which valid transactions are committed, and transaction latency, meaning the amount of time for the effect of a transaction to be acknowledged by other network nodes.

B. *Experimental Results*

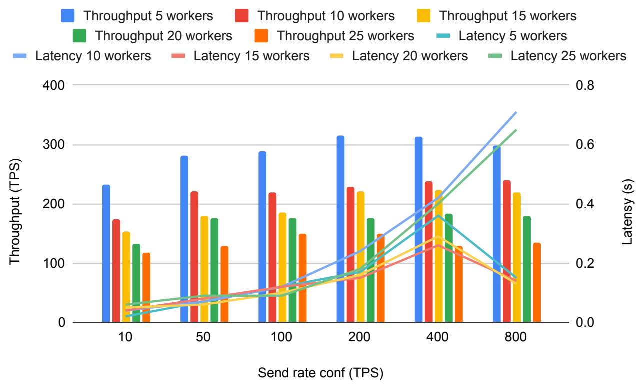
1) Create Operations: For the create operations, we will use the *Submit Auction* smart contract function. The results in Figure 4a show that the throughput increases as the send rate increases. We can also observe that fewer workers generally lead to higher throughputs. The throughput is quite high, allowing for over 60 transactions per second in all experimental runs at some point. Furthermore, this is without any failed transactions. The average latency increases in accordance with the send rate, and the number of workers does not appear to affect this.

2) Read Operations: For the read operations, we will use the *Query Auction* smart contract function. The results in Figure 4b indicate that the throughput is higher if there are fewer workers. However, the high throughput which allows for over 150 TPS in all experimental runs indicates the system is scalable. The throughput tends to increase until the send rate is about 200 TPS, at which point it tends to stagnate or decrease. Regarding the average latency, we noticed an interesting result, namely, it tends to increase as the send rate increases. However, sometimes, the latency begins to decrease at very high send rates. This behavior does not correlate to the number of workers, so this may be simply a glitch. Comparing the results of the read operations to that of the create operations, we observe a significantly higher throughput and significantly lower latency.

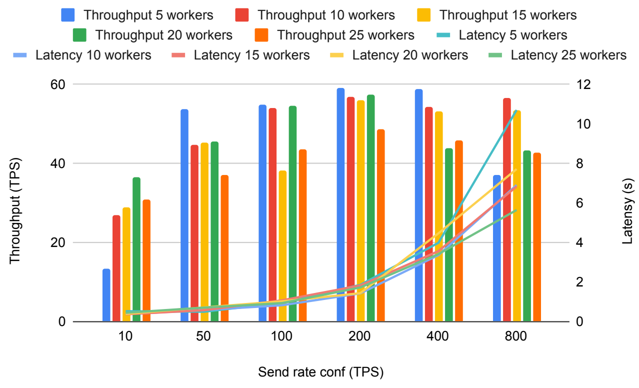
3) Update Operations: For the update operations, we use the *Submit Bid* smart contract function. From the results in Figure 4c we can see that the throughput tends to increase with the send rate, although if more workers are used it begins to decrease at higher send rates. In terms of the throughput, the high possible number of transactions per second, which approaches 60 TPS, indicates that the system is scalable. The latency results indicate that it increases in accordance with the send rate, similarly to create operations.



(a) Create operations: throughput and latency



(b) Read operations: throughput and latency



(c) Update operations: throughput and latency

Fig. 4. The performance of the dApp smart contracts.

V. Related Work

Current related research is in multiple directions. Decentralized auctions on the blockchain have been proposed to have wide potential [17]. Prasad et al. [18] developed a decentralized marketplace on the Ethereum blockchain. Their motivation for this is that centralized platforms can block merchants from participating at will, the fees paid to the platform when listing and selling a product, and the lack of privacy of user data. Pop et al. [19] designed an Ethereum-based implementation of English, Dutch and first-price sealed-bid auctions. They conclude that their solution offers a more secure alternative to traditional online auctioning systems by providing a stand-alone platform that does not require separate payment systems. Dolenc et al. [20] provide an overview of contemporary dApp blockchain development frameworks. The authors surveyed several blockchain platforms, e.g., Ethereum, Hyperledger, Hashgraph, EOS, Corda, IOTA, and Multichain. They compare the major properties of blockchain technologies, such as ledger type, consensus mechanisms, network types, and others. Furthermore, they compare the performance of blockchain systems by examining their throughput and latency. In our previous work [9], we introduced a framework called AWESOME to build a decentralized cloud marketplace and to address the challenges, e.g., service provider selection and service quality assurance. However, the proposed model is still in the conceptual design stage and lacks usability enhancements for user interaction. To the best of our knowledge, this paper is the first model that provides a customizable and flexible GUI module for users in a decentralized service marketplace.

VI. Conclusion

This paper proposes a customizable dApp framework for user interactions in decentralized service marketplaces. Although the intention was to develop this framework for cloud marketplaces specifically, the software produced is agnostic enough that it can be used for other kinds of marketplaces, such as data marketplaces. This is because any asset or service can be defined within the framework. Furthermore, although a commercial product was not developed, decentralized marketplaces are of growing interest in academia and industry. Because the software is an open-source project, researchers and software developers can use the project for their academic or industrial purposes to develop their own decentralized marketplaces.

For our future work, more blockchain technologies (e.g., Quorum, Multichain, Corda) and auction models (e.g., VCG auction, double auction) can be considered to integrate with the current framework. In addition, advanced algorithms can be designed to optimize the witness module further.

Acknowledgment

This research is funded by the European Union's Horizon 2020 research and innovation program under grant agreements 825134 (ARTICONF project), 862409 (BlueCloud project) and 824068 (ENVRI-FAIR project). The research is also supported by the Chinese Scholarship Council, and EU LifeWatch ERIC.

References

[1] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. Leung, "Decentralized applications: The blockchain-empowered software system," *IEEE Access*, vol. 6, pp. 53 019–53 033, 2018.

[2] L. Teixeira, I. Amorim, A. U. Silva, J. a. C. Lopes, and V. Filipe, "A new approach to crowd journalism using a blockchain-based infrastructure," in *Proceedings of the 18th International Conference on Advances in Mobile Computing & Multimedia*, ser. MoMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 170–178.

[3] N. Karandikar, R. Abhishek, N. Saurabh, Z. Zhao, A. Lercher, N. Marina, R. Prodan, C. Rong, and A. Chakravorty, "Blockchain-based prosumer incentivization for peak mitigation through temporal aggregation and contextual clustering," *Blockchain: Research and Applications*, vol. 2, no. 2, p. 100016, 2021.

[4] N. Saurabh, C. Rubia, A. Palanisamy, S. Koulouzis, M. Sefidanoski, A. Chakravorty, Z. Zhao, A. Karadimce, and R. Prodan, "The articonf approach to decentralized car-sharing," *Blockchain: Research and Applications*, vol. 2, no. 3, p. 100013, 2021.

[5] Z. Shi, H. Zhou, C. de Laat, and Z. Zhao, "A bayesian game-enhanced auction model for federated cloud services using blockchain," *Future Generation Computer Systems*, vol. 136, pp. 49–66, 2022.

[6] K. Wu, "An empirical study of blockchain-based decentralized applications," *arXiv preprint arXiv:1902.04969*, 2019.

[7] H. Zhou, Z. Shi, X. Ouyang, and Z. Zhao, "Building a blockchain-based decentralized ecosystem for cloud and edge computing: an ALLSTAR approach and empirical study," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 6, pp. 3578–3594, 2021.

[8] Z. Shi, H. Zhou, Y. Hu, S. Jayachander, C. de Laat, and Z. Zhao, "Operating Permissioned Blockchain in Clouds: A Performance Study of Hyperledger Sawtooth," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. Amsterdam, Netherlands: IEEE, Jun. 2019, pp. 50–57.

[9] Z. Shi, S. Farshidi, H. Zhou, and Z. Zhao, "An auction and witness enhanced trustworthy sla model for decentralized cloud marketplaces," in *2021 ACM International Conference on Information Technology for Social Good (GoodIT)*. ACM, 2021, pp. 109–114.

[10] L. Sun, J. Singh, and O. K. Hussain, "Service level agreement (sla) assurance for cloud services: A survey from a transactional risk perspective," in *Proceedings of the 2012 International Conference on Advances in Mobile Computing & Multimedia*. ACM, 2012, pp. 263–266.

[11] R. Mühlberger, S. Bachhofner, E. C. Ferrer, C. Di Ciccio, I. Weber, M. Wöhrer, and U. Zdun, "Foundational oracle patterns: Connecting blockchain to the off-chain world," in *2020 International Conference on Business Process Management*. Springer, 2020, pp. 35–51.

[12] C. Liu and Z. Li, "Comparison of centralized and peer-to-peer decentralized market designs for community markets," *IEEE Transactions on Industry Applications*, vol. 58, no. 1, pp. 67–77, 2022.

[13] J. Li, A. Grintsvayg, J. Kauffman, and C. Fleming, "Lbry: A blockchain-based decentralized digital content marketplace," in *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, 2020, pp. 42–51.

[14] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018, pp. 90–97.

[15] C. Dai, *DEX: A DApp for the Decentralized Marketplace*. Singapore: Springer Singapore, 2020, pp. 95–106.

[16] R. B. Uriarte, H. Zhou, K. Kritikos, Z. Shi, Z. Zhao, and R. De Nicola, "Distributed service-level agreement management with smart contracts and blockchain," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 14, Jul. 2021.

[17] Z. Shi, C. de Laat, P. Grosso, and Z. Zhao, "When blockchain meets auction models: A survey, some applications, and challenges," 2021, available at arXiv 2110.12534.

[18] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018, pp. 90–97.

[19] C. Pop, M. Prata, M. Antal, T. Cioara, I. Anghel, and I. Salomie, "An ethereum-based implementation of english, dutch and first-price sealed-bid auctions," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2020, pp. 491–497.

[20] D. Dolenc, J. Turk, and M. Pustišek, "Distributed ledger technologies for iot and business dapps," in *2020 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*, 2020, pp. 1–8.