Enhancing Blockchain-based Processes with Decentralized Oracles

Davide Basile[1 [0000−0002−5804−4036]], Valerio Goretti[1 [0000−0001−9714−4278]], Claudio Di Ciccio[1 [0000−0001−5570−0475]], and Sabrina Kirrane[2 [0000−0002−6955−7718]]

[1] Sapienza University of Rome, Italy
{basile.1810355,goretti.1811110}@studenti.uniroma1.it,
claudio.diciccio@uniroma1.it
[2] Vienna University of Economics and Business, Austria
sabrina.kirrane@wu.ac.at

Abstract. The automation of business processes via blockchain-based systems allows for trust, reliability and accountability of execution. The link that connects modules that operate within the on-chain sphere and the off-chain world is key as processes often involve the handling of physical entities and external services. The components that create that link are named oracles. Numerous studies on oracles and their implementations are arising in the literature. Nevertheless, their availability, integrity and trust could be undermined if centralized architectures are adopted, as taking over an oracle could produce the effect of a supply-chain attack on the whole system. Solutions are emerging that overcome this issue by turning the architecture underneath the oracles into a distributed one. In this paper, we investigate the design and application of oracles, distinguishing their adoption for the in-flow or out-flow of information and according to the initiator of the exchange (hence, pull- or push-based).

Keywords: Decentralized applications; Business process management; Distributed architectures

1 Introduction

Since its inception, the technologies related to the blockchain world are constantly evolving. In particular, its decentralized aspect has offered a development environment for Decentralized Applications (DApp), where data integrity and consistency are crucial factors [12]. However, applications developed on such platforms are unable to obtain information from the off-chain world, and cannot directly alter the outer world status [5,3]. Therefore, intermediate components named oracles have been introduced to open up the blockchain to the real world [25,18].

One of the usages in which DApps have shown potential is the coordination of business processes between multiple parties [16,8]. Especially in this scenario, oracles represent the trusted link with external sources of information. The possibility of erroneous or counterfeit information can result in major financial implications for the various stakeholders [23,13]. By distributing and decentralizing the transmitted information, and using redundancy procedures, the likelihood of such problems is greatly

reduced – which is also one of the driving factors at the core of blockchains. Indeed, decentralization increases the robustness and security of transmission operations by removing the problems associated with a single point of failure [11].

This paper studies the effect of decentralizing blockchain oracle architectures in terms of availability, integrity and trust. In particular, we examine the design and the implementation of decentralized and centralized oracles for the Ethereum platform, categorized as per the patters described in [18]. Unlike the typical scenario for oracles, we consider cases in which different off-chain sources retain separate parts of an information to be collected, or separate targets receive information from the blockchain. The proposed implementations are then evaluated in terms of both latency and costs.

The remainder of the paper is structured as follows. Section 2 presents the necessary background in terms of blockain platforms, the Ethereum ecosystem and the role of blockchain oracles. Section 3 introduces the motivating use case scenario used to guide our work. Section 4 sketches our blockchain oracle conceptual framework. Section 5 provides and overview of our performance evaluation, while Section 6 identifies open challenges and opportunities. Finally, we present our conclusions and plans for future work in Section 7.

2 Background

In the following, we briefly present background information on blockchains, with a special focus on the Ethereum ecosystem, and blockchain oracles.

2.1 Blockchain: Definition and Applications

A blockchain is a protocol for the distributed management of a data structure in which transaction are stored sequentially in an append-only list (the ledger). Updates on the ledger are communicated via sequential blocks that are built and validated (i.e., mined), and then broadcasted among the nodes in the network. The ledger is replicated in all nodes of the network. Nodes agree on the inclusion of the next block information via consensus algorithms [26]. Its decentralized, persistent and immutable characteristics make blockchain suitable for the needs of automated systems in which interactions between multiple untrusted parties are recorded [10]. Such systems have long been primarily used for payments via cryptocurrency transactions, as their infrastructure allows for the storage and regulation of exchanges without the arbitration of external authoritative entities [19].

With the advent of Ethereum [5], second-generation blockchain platforms emerged as the blockchain turned from being mainly an e-cash distributed management system to a distributed programming platform at the basis of Decentralized Applications (DApps) [17]. In particular, Ethereum enabled the deployment and run of smart contracts (i.e., stateful software artefacts exposing variables and callable methods) in the blockchain environment through the Ethereum Virtual Machine (EVM). The code of the deployed smart contract is stored into the blockchain itself. Every time a user interacts with a smart contract method, a new transaction is generated. As the code is executed by the EVM and not locally, users are required to pay fees (the so-called “gas”) as a

Table 1: Classification of oracles [18]

Inform. flow direction		Information exchange initiator	
		Pull	Push
	In	An on-chain component requires information from the outside word	An off-chain component initializes the procedure and sends data from outside the blockchain
	Out	An off-chain component requires information from the blockchain	An on-chain component initializes the procedure and sends data from the blockchain to the outside world

compensation for the computational power used. The users can specify the maximum limit they would pay and the price per gas unit in terms of the Ethereum native cryptocurrency (Ether). The amount of gas to be paid is proportional to the complexity of the code and the operations involved.

Notice that smart contracts can be invoked from the off-chain and, during the method execution, exchange messages within the on-chain spheres with other smart contracts. However, on-chain code cannot directly invoke off-chain programs for the sake of consistency and determinism. Ethereum smart contracts can emit so-called events [7], namely developer-specified data fields included within transactions that typically mark relevant stages of the execution. Off-chain software artefacts can subscribe to such events to react to the signalled statuses.

The new capabilities unlocked an array of new application domains for blockchains, including sectors like insurance and music and areas such as the internet of things and cybersecurity [1,20]. As emphasized by Tareq Ahram et al. [23], a key application domain for blockchains is supply chain management. In this scenario, blockchains are used to record the data generated in every step of the supply chain, by creating an immutable history of the good produced or the service delivered. In this way, the blockchain can greatly facilitate the recording of assets and the tracking of invoices, payments and orders. The motivating use case scenario in Section 3 is rooted in this domain.

2.2 Blockchain Oracles

The variety of DApps developed on the Ethereum blockchain has underlined the need to ensure the robustness, consistency and persistence of blockchain data by defining a structural context in which this technology is proposed as a closed and self-contained system unable to communicate with the outside world [17,25]. The inability of smart contracts to access data that are not already stored on-chain can be a limiting factor for many application scenarios such as that of multi-party business processes. The solution to this problem comes in the form of oracles [24]. An oracle can be seen as a bridge that allows for the communication between the on-chain and the off-chain world. The DApp should be able to trust the oracle in the same way as it does so with the information from within the blockchain. Reliability for oracles is key [15,2]. Moreover, an oracle has the arduous task of acting as a link between the blockchain application and different external entities, which may potentially be characterized by different technologies and mechanisms. Therefore, the designation and sharing of a well-defined protocol becomes fundamental for the proper functioning of the service. Mühlberger et al. [18]

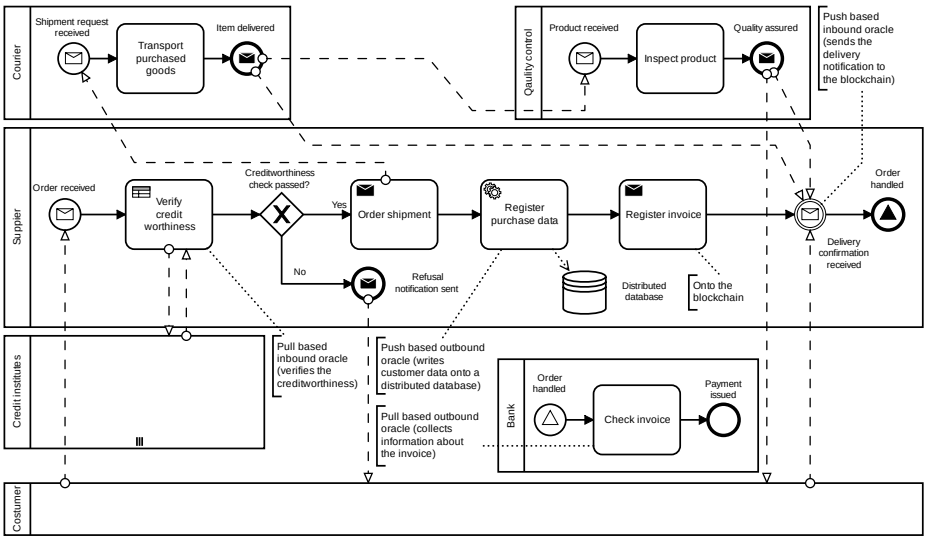

Fig. 1: BPMN diagram of the use case scenario

describe oracle patterns that consider two dimensions: the information direction (inbound or outbound) and the initiator of the information exchange (pull- or push-based). Inbound oracles inject data into the blockchain from the outside, whereas outbound oracles transmit information from the blockchain to the outside. Pull-based oracles are such that the initiator is the recipient of the information, whereas with push-based oracles the initiator is the sender of the information. By combining the push-/pull-based and inbound/outbound classifications, they define four oracle design patterns. Table 1 summarizes these types of oracles: The pull-based inbound oracle (henceforth, *pull-in* oracle for simplicity) is used when an on-chain component starts the procedure and injects data from the real world. The push-based inbound (*push-in* for short) oracle is used by an off-chain component to send data to the blockchain. The pull-based outbound (*pull-out*) oracle is used when an off-chain component needs to retrieve data from the blockchain. Finally, the push-based outbound (*push-out*) oracle allows an on-chain component to transmit information outside the blockchain. In addition to the information direction, Beniiche [3] categorizes existing oracle solutions according to the source of information (human, software or hardware) and on their centralized or decentralized architecture. In this paper, we are interested in the design and use of decentralized oracles that realize either of the above-mentioned four patterns in the context of a blockchain-based process execution.

3 Motivating Use Case Scenario

Figure 1 illustrates a multi-party order-to-cash business process involving a supply chain depicted as a BPMN collaboration diagram [9]. We will use this process throughout the paper as a running example and pinpoint the employment of decentralized ora-

cles. We recall that, according to the classification of Mühlberger et al. [18], oracles are categorized as inbound or outbound, according to the direction of the information flow, and as pull-based or push-based, based on the initiator of the information exchange.

In the first part of the workflow, the supplier wants to verify the creditworthiness of the customer. This verification is based on the usage of a pull-in oracle. If the verification generates a positive result, the supplier places the order and it orders the shipment. If the verification fails, the order is refused. The decentralized architecture of the oracle allows for the retrieval of distributed information about the creditworthiness, as the customer has open accounts in multiple credit institutions. We assume the sensitive information about the customer to be properly protected from malicious treatment or leakage through the usage of existing privacy-preserving record-linkage techniques [21]. Once the order is placed, and the product is handed to the courier, the shipment procedure starts. In the meantime, the supplier records the data of the purchase order into an external distributed database via a push-out oracle. The decentralized architecture of the oracle fits with the need to send data to a destination consisting of multiple nodes as the distributed database, in our scenario, resorts to physical instances. After that, the supplier registers the invoice in the blockchain. Meanwhile, the courier delivers the ordered product. At the customer's side, a quality control specialist checks that the consigned goods conform with the standards. If so, a push-in oracle uses the blockchain as a notarization means to record that the delivery succeeded. Notice that this passage requires three actors to give their confirmation based on three distinct information bits: the courier (for the consignment), the quality control specialist (for the status of the consigned material) and the customer (for the receipt of the goods). The push-in oracle is thus decentralized as well, as it requires a confirmation from multiple parties. Finally, the banking system can unlock the payment. In response to the notification of the finalized handling of the order, the bank verifies that the invoice is stored on the blockchain. As a successive layer of security, it retrieves the data from multiple, physically distinct blockchain nodes – thereby employing a decentralized pull-based oracle.

In the following section, we show a possible reference architecture for the decentralized version of the aforementioned oracle categories.

4 Decentralized Oracles

The main limitation that characterizes centralized oracles is the presence of a unique operative unit that works in order to make the information flow between the blockchain and the outer environment. This particular aspect can cause several critical issues that put the entire production chain at risk. The first one is the problem of possibly having a single point of failure as the oracle could be the weak link for cyber attacks. It is interesting to notice that attacks in which a trusted software component is injected with malicious code fall under the name of (software) supply-chain attacks [22]. Indeed, since the architecture provides only one operative unit, a potential malfunction determines the end of (trusted) communication and the potential loss of availability. Moreover, the centralization requires a greater guarantee of correctness of the transmitted information. The single component responsible for the communication cannot tolerate wrong or incoherent data.

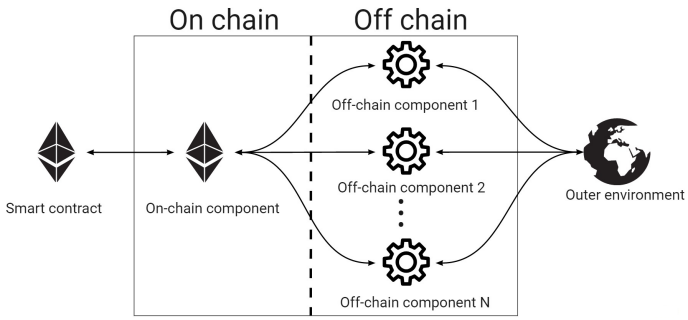

Fig. 2: Decentralized oracles architecture at large

Decentralized oracles, instead, resort to multiple independent components that send and receive information. In this case, the oracle becomes an information distribution network regulated by internal protocols such as an own consensus mechanism or incentivization strategies [14]. It is possible to adopt different approaches to the question of oracles' consensus. For example, some systems already on the market such as Gnosis[3] and Augur[4] adopt a voting mechanism combined with human oracles. Other systems, such as Chainlink,[5] propose a fully automated majority approach. The details on the management of those networks go beyond the scope of the paper. We refer to [4] for an overview of the mechanisms underlying an envisioned decentralized oracle network.

Without loss of generality, we assume here a fair behavior of the oracles and an inner consensus algorithm based on the agreement of the totality of the involved components. In the remainder, we will show how decentralization can be applied to oracles.

4.1 Architecture Overview

Regardless of the type under consideration, oracles can be split into two main tiers, as illustrated in Fig. 2. The on-chain tier manages the interaction between the oracle system and the on-chain world. It has a single software component inside, namely a full-fledged smart contract that can be seen as an entry point for decentralized applications that want to use that specific oracle system. The ways the DApp interacts with the on-chain tier is defined by the interaction protocol of the oracle itself. The off-chain tier is used to manage the interaction between the real world and the oracle. The two components of the oracle are able to communicate by sending data to each other. In the Ethereum ecosystem, when the on-chain tier sends data to the off-chain tier, it generates a new event containing the relevant information, which is caught in the off-chain tier. The off-chain tier can send data to the on-chain tier by using the methods exposed by the smart contract of the latter via a transaction with the necessary input. The decentralisation of the architecture takes place inside the off-chain tier, as multiple external

[3] https://www.gnosis.io/ Accessed: July 14, 2021.
[4] https://augur.net/ Accessed: July 14, 2021.
[5] https://chain.link/ Accessed: July 14, 2021.

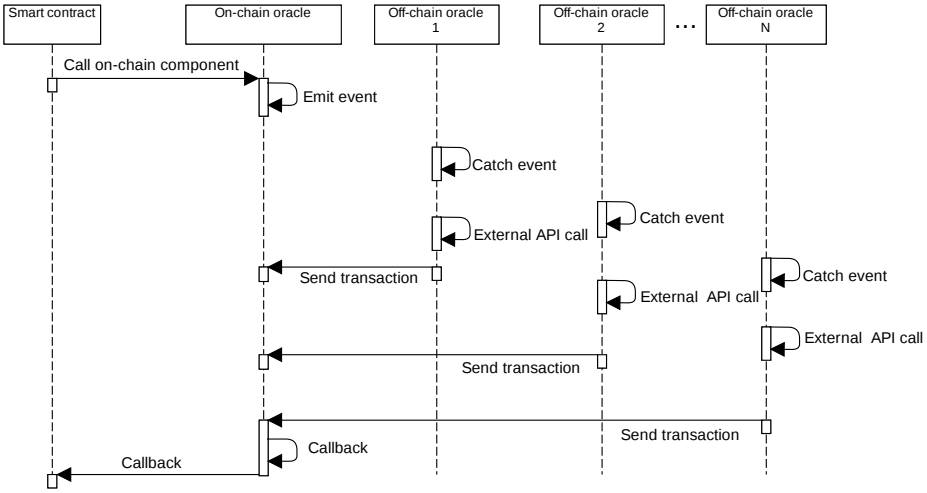

Fig. 3: Sequence diagram of the interactions with a decentralized pull-in oracle

modules interact with the on-chain world and operate independently, in order to retrieve data from the off-chain environment or send data to it.

In the following, we detail the design of our decentralized oracle architecture for each of the four categories described in [18]. We will refer to the oracles' on-chain components as on-chain oracles and to the components inside the off-chain tier as off-chain oracles for the sake of brevity.

4.2 Decentralized Pull-in Oracle

In a pull-in oracle, the interaction begins with the call from the smart contract implementing the process logic to the on-chain oracle, as depicted in the sequence diagram in Fig. 3. Considering the running example of Section 3, the purpose of the pull-in oracle is to connect the decentralized application with multiple credit institutions, in order to verify the creditworthiness of the customer – which is confirmed only if all credit institutes agree. The smart contract running the check activity interacts with the on-chain component of the oracle, generating a new request for verification. The on-chain component, then, emits a new event containing the data to be processed by the off-chain oracle (e.g., the customer personal information). At that point, the off-chain oracles catch the emission of the event, and they execute their business logic (e.g., the creditworthiness verification) based on different data sources via dedicated API calls (the credit institutions). Once the off-chain oracles have obtained the result of their computation, they invoke the on-chain oracle callback method to return the answer via transactions. In our simplified consensus mechanism, we assume that when all the off-chain components have sent their answer to the on-chain smart contract, it uses the callback method of the decentralized application, in turn, to return the aggregate result.

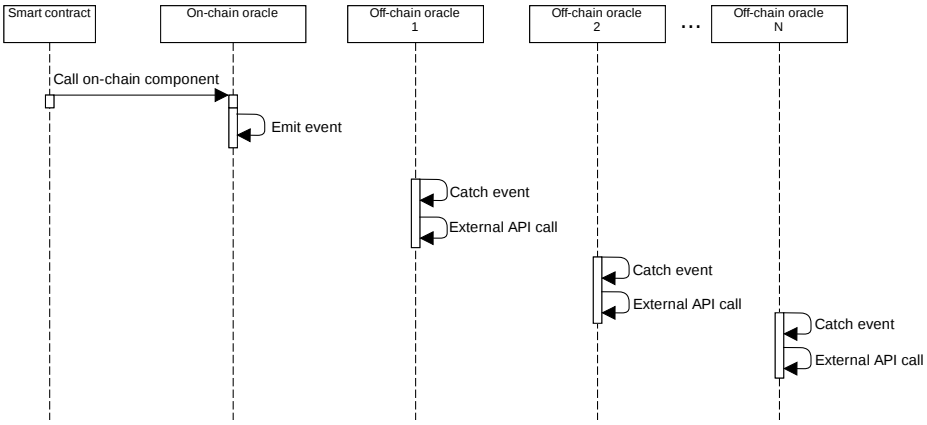

Fig. 4: Sequence diagram of the interactions with a decentralized push-out oracle

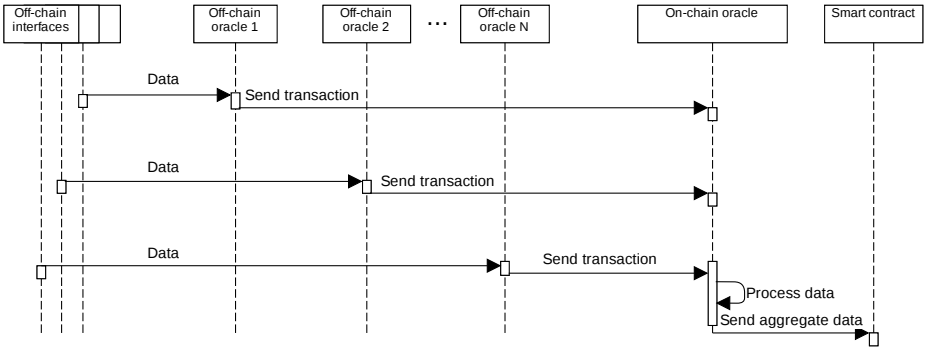

Fig. 5: Sequence diagram of the interactions with a decentralized push-in oracle

4.3 Decentralized Push-out Oracle

Figure 4 illustrates the interactions that realize the information exchange via a decentralized push-out oracle. Unlike the pull-in oracle, the source of information lies within the blockchain, which is by its nature a decentralized system. The procedure for pushing out the information starts when the smart contract creates a new request for the outbound transfer of data. The on-chain oracle generates a new event that contains the data to be exposed. When the off-chain components catch the event, they all operate independently, invoking external APIs. In our scenario, the purchase order data are stored in an external distributed database. In order to update the database with new orders, the smart contract underpinning the activity execution employs a push-out oracle and every off-chain oracle interacts with a different instance of the distributed database.

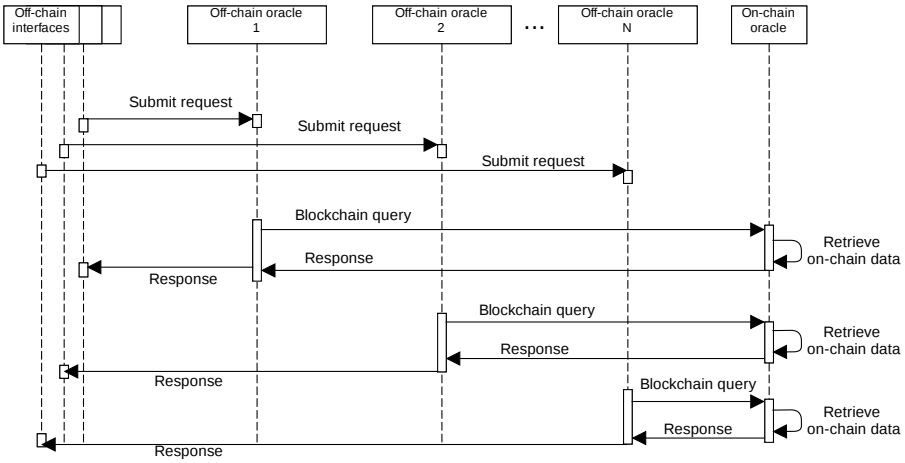

Fig. 6: Sequence diagram of the interaction with a decentralized pull-out oracle

4.4 Decentralized Push-in Oracle

Figure 5 depicts the transfer of information into a blockchain mediated by a decentralized push-in oracle. We assume an off-chain interface gathers data from various sources and sends it to the off-chain oracles. In turn, the off-chain oracles send the transaction with those data to the on-chain oracle, which is responsible for the collection of the different pieces of information, their aggregation and final communication with the smart contract. In our scenario, a decentralized push-in oracle is used to confirm that the delivery was successful, upon the notification from three different off-chain information providers, namely the carrier, the customer and the quality control specialist. Each of those information providers would interact with a dedicated off-chain oracle and a positive input from all of them would trigger the successful delivery confirmation to the smart contract.

4.5 Decentralized Pull-out Oracle

A decentralized pull-out oracle allows multiple external systems to retrieve on-chain information whenever required, as illustrated in Fig. 6. Without loss of generality, we assume the external systems to be collectively represented by an off-chain interface for the sake of readability, as in the case of push-in oracles. Our running example employs a decentralized pull-out oracle when the banking system retrieves the data related to the invoice, stored in the blockchain. A centralized oracle would obtain the information to return to the bank by accessing the blockchain through a single node. If that particular node was on an incoherent or corrupted fork of the blockchain, the retrieved information could be misleading.

The decentralized version of the oracle is used here to overcome the potential inconsistency of the blockchain data through its own decentralized nature, i.e., by resorting to

several independent components that watch the blockchain via different nodes. The process starts when the off-chain interface (invoked, e.g., by the banking system) requests data (e.g., the invoice information) to multiple off-chain oracles. Each of them generates a new query towards different nodes of the blockchain. In every node, the on-chain oracle would return the current response based on the local view of the blockchain, in turn given back to the requesting off-chain interface.

5 Implementation

In this section, we briefly describe a proof-of-concept prototype implementing the decentralized oracle architectures, and report on the experiments we conducted with it to have a preliminary assesment of its performance in terms of execution costs and latency.

5.1 Prototype and Experimental Setting

We implemented our system based on the Ethereum blockchain. We encoded the on-chain components of our prototype in Solidity, the most used language for Ethereum smart contracts at present. We resorted to Node.js scripts to implement the off-chain components and the Web3 library to let them interact with the blockchain, i.e., for the subscription to event emissions and to send transactions to the blockchain. The produced code is openly available on GitHub.[6]

To run our tests, we deployed the on-chain components of our prototype on Ropsten,[7] an Ethereum public testnet, in order to execute the tests and obtain the needed information about latency and costs. The test phase took place through four different accounts used to deploy the smart contracts and send transactions from the off-chain components. The transactions involved in our experiments are identified by the interactions with the following contracts and can be retrieved via Etherscan: `0xd7c351Eb1DfaFCf19bf47D3fe55a9D761a274bd7`; `0xA6a80830855c81b472A6aa9efb36bBA0fF36A5e4`; `0x7Cc2d01fb411b9E59924f2Bc79002f93E9A44ddB`; `0xAF69860c860A00d723fc0651f22637aF3b1B0d6D`.

5.2 Performance Tests

Using our proof-of-concept implementation, we have conducted a preliminary assessment of its performance in terms of latency and costs, in an attempt to have a rough estimation of the differences between centralized and decentralized oracle architectures. A fully-fledged comparative study is out of scope for this paper and we envision it as a relevant aim for future studies.

The first important consideration that we made is about the outbound (pull and push) oracles. Although every on-chain computation requires the triggering of a transaction,

[6] The implemented prototypes of the oracles used in the experiments are available at: https://github.com/DavideBasile1810355/Decentralized_Oracles/.

[7] Rospten explorer: https://ropsten.etherscan.io/. Accessed: July 14, 2021.

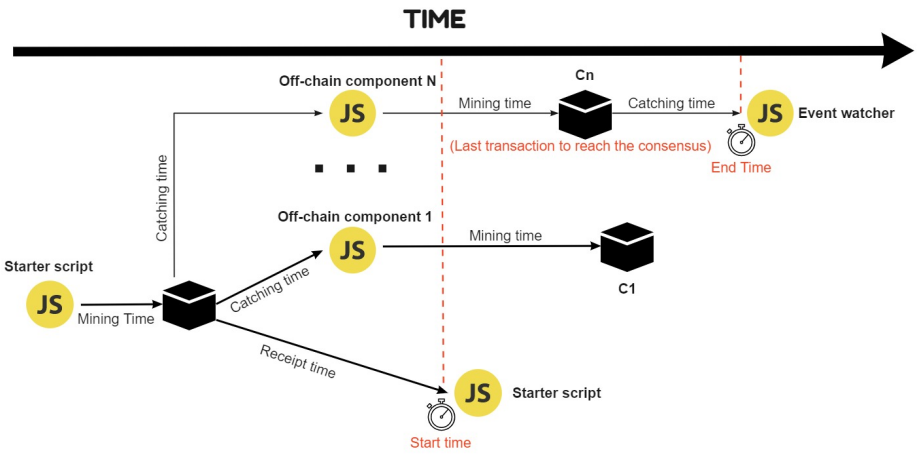

Fig. 7: Latency measurement for the pull-in oracle

we do not consider that transaction when measuring the performance of outbound oracles as they would not directly pertain to the oracle operations per se but rather to the pre-processing by the smart contract. Indeed, on-chain computation may be required to produce the data later retrieved by pull-out oracles, or for the production of the information to be transmitted off-chain by push-out oracles. However, from an abstract standpoint, this would depend on the kind of data treatments required rather than on the information exchange per se. In both cases, data is obtained by the off-chain components by catching the emission of an event, and this action has no cost for the oracle system. This aspect has two important consequences. First of all, interactions with outbound oracles do not necessarily involve any expenditure of gas. Furthermore, the transaction latency for these two kinds of oracles is irrelevant. The blockchain ecosystem does not affect in any way the global latency as no block mining is involved. However, we remark that blockchain is a distributed system and, as such, latency may occur from the information distribution itself within the network, aside from the block time or transaction latency. This is a crucial factor to consider for process-aware system designers implementing the operations on-chain: especially if numerous software components are involved, variable delays and possibly time-outs could affect the overall stability of the system.

The inbound implementations can provide interesting quantitative information that can be used for a preliminary performance assessment about costs and latency. We quantify the spending of the oracles in gas units and its equivalent amount in Euros. The exchange rate considered at the time of the experiments is 495 Euros per Ether (ETH), while the average gas price considered for the ETH/gas conversion is 8 Gwei ($0.000{,}000{,}008$ ETH) per gas unit. Regarding the experiments on latency, our goal was to measure the time elapsed between the event that starts the interaction and the arrival at destination (i.e., the blockchain) of the information.

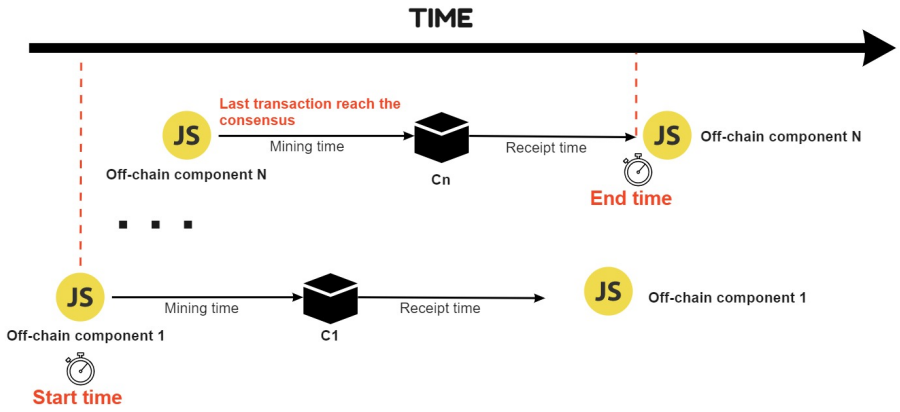

Fig. 8: Latency measurement for the decentralized push-in oracle

Figure 7 illustrates our time measurement scheme for the pull-in oracle. Considering our running example, the starter script represents the supplier's request to verify the creditworthiness of the customer. The pull-in oracle begins its execution as soon as the starter script receives the mining receipt of its request – then we start the timer. At that point, the off-chain component is activated by the on-chain component, and after it has retrieved the requested data from the off-chain environment, it sends a transaction to the on-chain oracle with that information. The end time of the measurement corresponds with the instant in which the on-chain component terminates the computation of the received input. In a decentralized scheme, oracles employ separate off-chain components that work independently. Therefore, the information processing from the on-chain component can begin only when the latest off-chain component has transacted its data.

Figure 8 depicts our measurement scheme for the decentralized push-in oracle (used in our running example for the delivery confirmation). The start time corresponds with the first transaction being sent by one of the off-chain components. The end time elapses when the latest confirmation receipt is received confirming the sending of aggregate data from the on-chain component. Notice that there is only one initial transaction and one final receipt in the centralized case.

Table 2 reports on the experimental results. For each test we executed 50 runs, totalling 200 runs (i.e., 100 for the centralized case and 100 for the decentralized case). Table 2(a) shows the results of the experiments for the centralized implementations, reporting the mean, minimum and maximum values, and the standard deviation. As it turns out, the fastest implementation is that of the pull-in oracle, with a mean latency value of 18.24 s and a standard deviation of 15.23 s. The push-in oracle, instead, took 23.54 s on average with a standard deviation of 16.56 s. For as far as costs are concerned, the most expensive implementation is that of the pull-in oracle (with a mean of 42,000.98 gas units, while the push-in oracle required 39,505.47 units).

Table 2: Latency and cost test results

(a) Centralized oracles

	Mean	Min	Max	Std. dev.
Pull-in oracle				
Cost (gas)	42,000.98	22,550	62,736	9039.54
Cost (euro)	0.17	0.09	0.25	/
Latency (seconds)	18.24	4.00	93.12	15.23
Push-in oracle				
Cost (gas)	39,505.47	38,003	42,239	2027.04
Cost (euro)	0.16	0.15	0.17	/
Latency (seconds)	23.54	4.00	72.92	16.56

(b) Decentralized oracles

	Mean	Min	Max	Std. dev.
Pull-in oracle				
Cost (gas), Node 1	69,232.97	25,919	94,816	24,124.19
Cost (gas), Node 2	60,565.05	22,919	94,794	27,895.94
Cost (gas), Node 3	49,377.23	25,919	109,522	27,000.39
Total cost (gas)	179,175.30	110,300	236,162	/
Total cost (euro)	0.72	0.44	0.95	/
Latency (seconds)	34.56	8.21	100.33	20.62
Push-in oracle				
Cost (gas), Node 1	42,737.50	30,098	58,720	7808.17
Cost (gas), Node 2	42,309	30,098	58,720	7846.29
Cost (gas), Node 3	30,620.50	30,098	58,720	8034.68
Total Cost (gas)	115,667	105,338	136,913	/
Total Cost (euro)	0.46	0.42	0.55	/
Latency (seconds)	27.78	5.79	69.65	14.34

We evaluated the execution cost for oracles both in terms of the singular off-chain components (which we denote as Node 1, Node 2, and Node 3) and in terms of the whole oracle system. Considering the motivating scenario, both the pull-in and the push-in oracles employ three off-chain components that work independently. The cost of the single interaction is given by the sum of all the transaction costs, generated by each independent component (denoted as "C1", ..., "Cn" in Figs. 7 and 8). Table 2(b) shows the results for the decentralized case. As it can be seen in the table, in both cases independent nodes of the same system determine different mean costs. The test shows that some nodes require on average more gas than others although they belong to the same oracle. In other words, the gas consumption of the three off-chain components is not balanced. This can be explained by the order whereby the off-chain nodes send their transaction to the on-chain component. Indeed, the on-chain component provides the decentralized application with the data when all the off-chain components have sent their transaction. By considering the single run, the last off-chain node that sends the transaction containing the data is the one that spends more. In this case, the code executed by the transaction has a higher computational complexity because it also includes the operations for the delivery of the data to the smart contract of the DApp. The slowest decentralized implementation is the pull-in oracle a mean of $34.56\,\mathrm{s}$, while the push-in oracle takes $27.78\,\mathrm{s}$ on average. Concerning the costs, the tests show that $179{,}175$ units of gas are spent for the pull-in oracle and $115{,}667$ units of gas for the push-in one.

6 Opportunities and Challenges

One of the main aspects of the proposed decentralization architecture is availability. By decentralizing the structure of the oracle we eliminate a single point of failure. In the case of the centralization, if the unique control entity in charge of the information flow fails, the entire system oracle stops working and the communication ends. On the

Table 3: Comparison table between centralized and decentralized implementations

	Centralized	Decentralized
Pull-in oracle		
Average cost (gas)	42,000.98	179,175.30
Average latency (seconds)	18.24	34.56
Push-in oracle		
Average cost (gas)	39,505.47	115,667
Average latency (seconds)	23.54	27.78

contrary, in the proposed decentralized architecture, there is no central authority, since every off-chain component interacts independently with the on-chain component, which is, in turn, deployed on a decentralized system (the blockchain). In this way, the risk of failure for the whole system is reduced.

Another aspect that is affected by the decentralization is the integrity of data. Multi-party processes that rely on oracles may need to perform complex operations involving significant amounts of resources, and they cannot tolerate faulty or altered data. The fact that the information is not maintained by only one entity decreases the risk of counterfeit data injection (as in the unlocking of funds upon the confirmation from multiple nodes that the invoice was registered). In this way, reliability and trust could be generated. Of course, it is necessary to define internal mechanisms so that an agreement between the different components can be achieved. This specific aspect can increase the complexity of the whole oracle system, thus the centralized version might be preferred in some scenarios.

The centralized and the decentralized implementations allowed us to carry out a preliminary analysis of the performance (latency and costs) of the different kinds of architecture for the inbound oracles. As illustrated in Table 3, in all cases the decentralized prototypes require higher costs and cause more latency than their centralized version. This can be explained by the presence of multiple transactions in the case of the decentralized versions. Indeed, our use case involves the definition of three off-chain components each of which generates one transaction for every procedure. Unlike the centralized versions that defines only one transaction for every information exchange. In this way, the mean cost of the system grows with the number of off-chain components involved. Alternatively, a decentralized system could check the agreement between the off-chain components in the real world, whereby only one transaction containing the final data would be generated. Therefore, it could serve as a viable alternative. However, if the entity in charge of sending the final transaction fails, the whole system stops working with such a solution, and the single point of failure problem persists.

Regarding latency, the difference between the two architectures is less evident. In the case of the pull-in oracle, the centralized version mean result is of $18.24\,\mathrm{s}$ against the $34.56\,\mathrm{s}$ of the decentralized version. The push-in centralized version, instead, generates a mean result of $23.54\,\mathrm{s}$, against the $27.78\,\mathrm{s}$ of the decentralized version.

7 Conclusion and Future Work

In this paper, we investigated on the use and development of decentralized oracles as a means to enhance availability, integrity and trust of information exchanges between the blockchain and the outer environment in a business process context. We started with the design and development of on-chain components that communicate with the off-chain modules developed in a centralized version. Subsequently, we turned the oracles architecture into a decentralized one and compared it with the previous version. Our prototype was evaluated in terms of execution costs and latency.

In this paper, we have focused on the Ethereum blockchain in particular. Our study will be complemented with the development of oracles that are compatible with other blockchain platforms and then study the use of decentralized oracles for communication between multiple blockchains. Furthermore, the Solid Web[8] has been recently proposed as a paradigm for web applications preserving data ownership and privacy. Reportedly, blockchain can be a key enabler of this novel paradigm [6]. Therefore, we will study the adoption of decentralized oracles to link decentralized systems and information producers and consumers to the Solid Web. Moreover, in this paper we have devised the merge and consistency-check of information exchanged with multiple off-chain components as an on-chain operation. Though more robust, this approach could incur higher costs than a purely off-chain mechanism. Therefore, an analysis of the best suitable trade-offs in terms of load-balancing and security of the two solutions is part of our envisioned future work. Finally, we will conduct in-depth studies on the scalability and robustness of the architecture, with an investigation on potential threats to security.

Acknowledgments. The authors are grateful to the reviewers for their precious feedback. The work of C. Di Ciccio was partially funded by the MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza and by the Sapienza research project "SPECTRA".

References

1. Ahram, T., Sargolzaei, A., Sargolzaei, S., Daniels, J., Amaba, B.: Blockchain technology innovations. In: 2017 IEEE Technology Engineering Management Conference (TEMSCON). pp. 137–141 (2017)
2. Al-Breiki, H., Rehman, M.H.U., Salah, K., Svetinovic, D.: Trustworthy blockchain oracles: Review, comparison, and open research challenges. IEEE Access **8**, 85675–85685 (2020)
3. Beniiche, A.: A study of blockchain oracles. CoRR **abs/2004.07140** (2020)
4. Breidenbach, L., Cachin, C., et al.: Chainlink 2.0: Next steps in the evolution of decentralized oracle networks (2021), https://research.chain.link/whitepaper-v2.pdf
5. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper **3**(37) (2014)
6. Cai, T., Yang, Z., Chen, W., Zheng, Z., Yu, Y.: A blockchain-assisted trust access authentication system for solid. IEEE Access **8**, 71605–71616 (2020)
7. Dannen, C.: Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. Apress, USA, 1st edn. (2017)

[8] Solid Web: https://solid.mit.edu. Accessed: July 14, 2021.

8. Di Ciccio, C., Cecconi, A., Dumas, M., García-Bañuelos, L., López-Pintado, O., Lu, Q., Mendling, J., Ponomarev, A., Tran, A.B., Weber, I.: Blockchain support for collaborative business processes. Inform. Spektrum **42**(3), 182–190 (2019)
9. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
10. Feig, E.: A framework for blockchain-based applications. CoRR **abs/1803.00892** (2018)
11. Hens, P., Snoeck, M., Backer, M.D., Poels, G.: Decentralized event-based orchestration. In: Business Process Management Workshops. pp. 695–706. Springer (2010)
12. Hu, Y., Liyanage, M., Manzoor, A., Thilakarathna, K., Jourjon, G., Seneviratne, A., Yliantila, M.: The use of smart contracts and challenges. CoRR **abs/1810.04699** (2018)
13. Ladleif, J., Weber, I., Weske, M.: External data monitoring using oracles in blockchain-based process execution. In: BPM (Blockchain and RPA Forum). pp. 67–81. Springer (2020)
14. Lo, S.K., Xu, X., Staples, M., Yao, L.: Reliability analysis for blockchain oracles. Computers & Electrical Engineering **83**, 106582 (2020)
15. Mammadzada, K., Iqbal, M., Milani, F., García-Bañuelos, L., Matulevicius, R.: Blockchain oracles: A framework for blockchain-based applications. In: BPM (Blockchain and RPA Forum). pp. 19–34. Springer (2020)
16. Mendling, J., Weber, I., van der Aalst, W.M.P., vom Brocke, J., Cabanillas, C., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Manag. Inf. Syst. **9**(1), 4:1–4:16 (2018)
17. Moantly, D.: Ethereum for architects and developers: With case studies and code samples in solidity. Apress (2018)
18. Mühlberger, R., Bachhofner, S., Ferrer, E.C., Di Ciccio, C., Weber, I., Wöhrer, M., Zdun, U.: Foundational oracle patterns: Connecting blockchain to the off-chain world. In: BPM (Blockchain and RPA Forum). pp. 35–51. Springer (2020)
19. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep. (2008)
20. Nofer, M., Gomber, P., Hinz, O., Schiereck, D.: Blockchain. Bus. Inf. Syst. Eng. **59**(3), 183–187 (2017)
21. Nóbrega, T., Pires, C.E.S., Nascimento, D.C.: Blockchain-based privacy-preserving record linkage: enhancing data privacy in an untrusted environment. Information Systems **102**, 101826 (2021)
22. Ohm, M., Plate, H., Sykosch, A., Meier, M.: Backstabber's knife collection: A review of open source software supply chain attacks. CoRR **abs/2005.09535** (2020)
23. Tijan, E., Aksentijevic, S., Ivanić, K., Jardas, M.: Blockchain technology implementation in logistics. Sustainability **11**, 1185 (02 2019)
24. Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A.B., Chen, S.: The blockchain as a software connector. In: WICSA. pp. 182–191. IEEE Computer Society (2016)
25. Xu, X., Pautasso, C., Zhu, L., Lu, Q., Weber, I.: A pattern collection for blockchain-based applications. In: EuroPLoP. pp. 3:1–3:20. ACM (2018)
26. Zheng, Z., Xie, S., Dai, H., Chen, X., Wang, H.: An overview of blockchain technology: Architecture, consensus, and future trends. In: BigData Congress. pp. 557–564. IEEE Computer Society (2017)