Journal of
Sensor and Actuator Networks

MDPI

Article

Veritas: Layer-2 Scaling Solution for Decentralized Oracles on Ethereum Blockchain with Reputation and Real-Time Considerations

Moustafa Mowaffak Saad, Dalia Sobhy and Amani A. Saad *

Computer Engineering Department, Arab Academy of Science and Technology and Maritime Transport, Alexandria 3650111, Egypt; mmowaffak@gmail.com (M.M.S.); dalia.sobhi@aast.edu (D.S.)
* Correspondence: amani.saad@aast.edu

Abstract: Blockchainsand smart contracts are pivotal in transforming interactions between systems and individuals, offering secure, immutable, and transparent trust-building mechanisms without central oversight. However, Smart Contracts face limitations due to their reliance on blockchain-contained data, a gap addressed by 'Oracles'. These bridges to external data sources introduce the 'Oracle problem', where maintaining blockchain-like security and transparency becomes vital to prevent data integrity issues. This paper presents *Veritas*, a novel decentralized oracle system leveraging a layer-2 scaling solution, enhancing smart contracts' efficiency and security on Ethereum blockchains. The proposed architecture, explored through simulation and experimental analyses, significantly reduces operational costs while maintaining robust security protocols. An innovative node selection process is also introduced to minimize the risk of malicious data entry, thereby reinforcing network security. *Veritas* offers a solution to the Oracle problem by aligning with blockchain principles of security and transparency, and demonstrates advancements in reducing operational costs and bolstering network integrity. While the study provides a promising direction, it also highlights potential areas for further exploration in blockchain technology and oracle system optimization.

Keywords: blockchain; oracle; gas; Ethereum; sidechain

Citation: Saad, M.M.; Sobhy, D.; Saad, A.A. Veritas: Layer-2 Scaling Solution for Decentralized Oracles on Ethereum Blockchain with Reputation and Real-Time Considerations. *J. Sens. Actuator Netw.* **2024**, *13*, 21. https://doi.org/10.3390/jsan13020021

Academic Editor: Lei Shu

Received: 31 December 2023
Revised: 14 February 2024
Accepted: 20 February 2024
Published: 7 March 2024

1. Introduction

Layer-2 scaling solutions are a way to make the Ethereum network more scalable, faster and cheaper [1]. They are a way to mitigate the effects of a 'main net' congestion when too many transactions are processed simultaneously [2]. The maximum throughput of 'Ethereum' at the moment is around 15 tx/s [3], putting it well below Visa's network, for example, which can handle upwards of 2000 tx/s. This solution has not been used before to try and mitigate the 'Oracle problem'. When a smart contract is executed, it is limited by the data available to it on the blockchain. In this case, the blockchain is treated as a black box, separated from the outside world, and that is by design to achieve the desired level of security, transparency, and reliability. This limits the applications available for smart contracts to only financial-related applications, but the potential is much greater.

This is solved by augmenting the blockchain with an 'Oracle' [4,5], essentially a bridge between the blockchain and the outside world. This oracle can fetch data into the blockchain or pass data from the blockchain to an outside consumer. Oracles are classified using many classification points, but we focus on the 'centralized' vs. 'decentralized' point of comparison [6,7]. Centralized Oracles are inherently insecure because they are a single point of failure. This contradicts the notion of a blockchain, which aims to eliminate central authorities and Single Points Of Failure (SPOF). If the Oracle is compromised, the whole blockchain is compromised; that is why it is called the 'Oracle problem'. Some solutions have been introduced to address this, some centralized and some decentralized, each with tradeoffs and compromises [8]. Layer-2-based solutions have not been seen before in

decentralized Oracle design. In this context, we aim to propose a novel approach that can be a viable, scalable, and effective solution to the problem.

To that end, we propose *Veritas*, a layer-2-based decentralized oracle for Ethereum-based smart contracts. With Veritas, the Oracle nodes are part of a sidechain network based on the same consensus rules as Ethereum (proof of stake as of 2022) [9,10]; this achieves a high level of security and transparency desired for decentralized Oracle systems. All jobs and activities performed by the nodes are orchestrated by a central smart contract living on the Veritas sidechain; this includes processing jobs, assigning jobs, receiving and aggregating results, and penalizing dishonest Oracle nodes.

This paper therefore attempts to answer the following research questions:

- **RQ1:** How can we fetch external information using decentralized oracles that do not compromise security but are also efficient concerning gas usage? We propose a novel *cost-effective method* for sourcing data from a distributed network by having the Oracle nodes themselves as part of a side chain, which reduces the cost of fetching data to the main chain because all data fetching and aggregation is conducted and sent back in a single transaction.
- **RQ2:** Can we fetch external data into the Ethereum network for real-time applications without paying very high gas prices? We propose a protocol whereby real-time jobs specify parameters when requesting jobs that only send real-time data points after crossing predetermined thresholds. This significantly reduces transaction costs for real-time monitoring jobs for the blockchain.
- **RQ3:** Can we improve upon node selection and onboarding to combat malicious behavior? We propose an enhanced onboarding protocol that requires nodes to stake and provides a measure of security against possible Sybil attacks by making the creation of malicious nodes costly. Node selection provides measures to ensure we achieve load balancing, reputation consideration, and a degree of randomness.

The remainder of this paper is structured as follows. Section 2 discusses the necessary background to understand the proposed method. Section 3 briefly summarises the relevant literature and motivates this paper's contribution. Section 4 describes the proposed system architecture. Section 5 presents the experimental evaluation and results of our approach. Section 6 further discusses the method and potential threats to validity. The paper concludes and provides future work in Section 7.

2. Background

In this section, we introduce the background to blockchains, Ethereum, smart contracts, and the need for oracles. We also take a look at existing solutions to the problem and examine their trade-offs.

2.1. Blockchain

Blockchain is a peer-to-peer network with a distributed ledger hosted on every node in the network [11,12]. Nodes are anonymous but can transact with each other in a trustless manner without the requirement of a central authority to guarantee honest operation between participants [8]. Blockchain was introduced as an integral part of a peer-to-peer cash system (Bitcoin). However, it has evolved into a more mature technology that can address more than just financial applications [13,14]. Transactions are packed into blocks that are added to the distributed ledger and propagated to the rest of the network. There is a necessary consensus mechanism to ensure that all parties agree on the current version of the distributed ledger [15,16]. Bitcoin uses a consensus method called POW (Proof of Work) [17], where after a node validates a transaction and wishes to add it to a block, it has to solve a cryptographic problem that requires a lot of computation. Solving this problem indicates that the node has performed the necessary work to validate this block and is rewarded for the effort (power consumed and time) put into validating the transaction and block. The reward is a cryptocurrency known as Bitcoin [15], and this process of solving the cryptographic puzzle and validating it is known as mining. As mentioned,

blockchains have evolved since their inception, and transactions now involve more than just the transfer of cryptocurrency between parties; they can now involve code execution, hence the name (smart contracts [18,19]), which is discussed in Section 2.2.1 where we explain how Ethereum and smart contracts operate.

Blockchains provide a high degree of security [8] since the interaction and transactions between parties are validated, packed into blocks, cryptographically linked to previous blocks, and appended to an immutable distributed ledger. They do not require a central actor to guarantee honest operation. Participants are anonymous and interact with the blockchain through private and public key pairs and wallets to store currency.

2.2. Ethereum and Smart Contracts

2.2.1. Ethereum

Ethereum is a decentralized and open-source blockchain with the added functionality of executing and maintaining smart contracts [20,21]. It is among the other blockchains that support smart contract functionality, but it is the most popular one in use today with a market cap of more than 400 billion dollars [22]. The main currency for transacting and receiving rewards is called ether. Like the Bitcoin blockchain, Ethereum's consensus algorithm was, until recently, based on the POW algorithm, where participating nodes had to solve cryptographic puzzles to earn a block reward. Many criticisms have since come to light regarding the POW's high-power requirements and slow operation. Another consensus system, POS (Proof Of Stake) [9,23], has been introduced and used by many other blockchains, including Ethereum. POS works by asking nodes who wish to participate in the validation process to 'stake' a certain amount of an asset (in this case, ether), after which a node is chosen randomly from a pool of nodes to verify and validate transactions and add them to the distributed ledger. Ethereum's key innovation is that it started to view transactions not just as the transfer of currency and the change in account balances, but also as the change in the 'stake' of state machines. This is achieved through the functionality of smart contracts (which is discussed in Section 2.2.2).

Every participant in the network has a pair of keys (private and public) from which an address is generated. This address is the only way to define an account or a node on the Ethereum network. They transact with their associated address whenever a participant wishes to transact with another participant. The private key guarantees access to the wallet associated with that account.

2.2.2. Smart Contracts

Smart contracts are pieces of code that live on the Ethereum blockchain [18,19]. Each smart contract has its address and is deployed to the blockchain-distributed ledger through a special kind of transaction. Once a contract is deployed, it is immutable and cannot be changed. The only way to change a contract is to build the capability to do so and assign specific account addresses the authority to do so. A decentralized, authority-free contract does not lend special privileges to particular accounts. Smart contracts are written in an object-oriented language called 'Solidity' among other languages for other platforms [24]. An externally owned account (EOA) may be called a smart contract, i.e., a wallet address or another smart contract.

A smart contract is formed by creating a transaction similar to the one that transfers currency. However, instead, the target is the address of the smart contract, and in a field called 'data' in the transaction, the function name required to be executed is added along with the required parameters. This transaction and method calling cause the smart contract function to 'run' as a normal program would on a computer. In contrast, smart contracts run on EVMs, or 'Ethereum Virtual Machines' [22,25].

2.2.3. EVMs and Gas

The EVM is a quasi-Turing-complete state machine [22]. It is 'quasi' because execution is limited to the amount of gas allocated, as explained in Section 2.2.4. So, when a party

initiates a transaction (a call to a smart contract method), an initial state lives on all nodes in the network (the current consensus on the ledger). When propagated through the network, the transaction causes the nodes to run that method locally on the EVM and produce a state change that should be consistent across all nodes. In that sense, one can think of the combination of the Ethereum blockchain and the EVM as a worldwide, decentralized computer, where each node executes the code, and all nodes can come to a consensus about what the state should look like after the execution process. The problem with this arrangement is the known 'halting problem' [26], where a dishonest actor can create a malicious transaction and take advantage of faulty logic on the contract to execute a function indefinitely, one that never halts. Hence, the concept of 'gas' is introduced.

2.2.4. Gas and Fees

Gas is a concept introduced to the Ethereum ecosystem to prevent the halting problem and quantitatively estimate the work performed to execute a particular transaction to reward miners [21,27]. It refers to the unit that measures the computational effort required to execute specific operations on the Ethereum Virtual Machine network.

Transactions that execute on a node's (or client's) machine require computational resources; each executed instruction consumes a fixed and predefined amount of 'gas'. Therefore, a transaction initiator must pay an amount named $Cost_{Transaction}$, composed of the Gas_{Amount} required multiplied by a Gas_{Price}, specified by the transaction initiator, as mentioned in Equation (1).

$$Cost_{Transaction} = Gas_{Amount} * Gas_{Price} \tag{1}$$

When a transaction executes on an EVM, each instruction deducts the corresponding Gas_{Amount} required for that operation. If the transaction is complete and the allocated *gas* is not exhausted, the remaining amount is refunded to the initiator, and the used gas fees are paid to the miner. We suppose that *gas* is entirely exhausted before the transaction execution is completed. In that case, the state is reverted as if the transaction never took place and the miners are awarded the fee they earned from the computation they performed. When a transaction is propagated on the network for miners to execute, validate and add to the blockchain, they can choose which transactions to add or execute, and most of the time they execute those with higher fees to earn higher rewards. That means that the higher the *gasPrice* set by the initiator, the faster their transaction is included in the next block. When many transactions are validated on the network, gas prices increase. Therefore, *transactionFee* rises substantially as initiators compete for validator resources to include their transactions in the upcoming blocks [22,25,27].

The $\text{GasLimit}_{\text{block}}$ refers to the maximum number of gas units allowed to fit in a single block (Equation (2)). So if the gas limit for an Ethereum-based network is x, then the sum of all gas used to execute the transactions grouped into the block must be less than or equal to x. $\text{GasLimit}_{\text{block}}$ refers to the maximum amount of gas allowed for the execution of all transactions within a block on the blockchain. $\text{Gas}_{\text{trans}_i}$ resembles the gas required for an individual transaction, where (i) is the transaction index ranging from 1 to n. n represents the total number of transactions to be added to the block. Each transaction has an associated gas cost $\text{Gas}_{\text{trans}_i}$.

$$\text{GasLimit}_{\text{block}} \geq \sum_{i=1}^{n} \text{Gas}_{\text{trans}_i} \tag{2}$$

The concepts behind EVM and gas are crucial for our proposed method (Section 4), as the rules governing the Ethereum blockchain are the same as those that govern the Veritas sidechain since it is based on Ethereum.

2.3. Web3

Web3 refers to the next generation of the Internet, which uses blockchain technology to create a decentralized and trustless ecosystem [28,29]. Unlike traditional web applications,

which rely on central authorities for data storage and processing, Web3 aims to empower users with more control over their data and interactions. It enables peer-to-peer transactions, smart contracts, and decentralized applications (dApps) on public blockchains, fostering transparency, security, and privacy [30,31]. With Web3, users can directly interact with blockchain networks, manage their digital assets, and participate in decentralized governance. It can potentially revolutionize various industries, including finance, supply chain, healthcare, and more, providing a framework for the building of decentralized applications that are resistant to censorship, immutable, and verifiable by design [32].

2.4. Oracles and 'The Oracle Problem'

As previously mentioned in Section 2.1, the blockchain is a collection of blocks containing cryptographically chained transactions [4,7]. The only available data are those which are inherently available in the blockchain state, and we are only able to process information that exists in that state. This is a crucial property of any blockchain, by extension including the Ethereum blockchain. Blockchains are referred to as 'Siloed' in the sense that they are isolated from the outside world because the transactions in a block represent a mutation in only the existing internal state. In the case of *Bitcoin*, the transactions represent a change in the state of Bitcoin holdings every account has by registering the transfer of Bitcoin from one account to another. In the case of Ethereum, the transactions represent a change in the state of the EVM. It is 'intrinsically deterministic' because EVMs on different nodes must execute the same code with the same input and come up with the same conclusion, which modifies the blockchain state. That means there cannot be any source of 'randomness', which becomes very important as we state our design. To realize the full potential of smart contracts and for them to be used for real-world applications, they need to be able to interact with the outside world. They must consume, produce, and process data outside the blockchain [18].

Since the only way to introduce entropy and extrinsic data into the blockchain is through the data payload in smart contract function calls, the only actors who can do so are transaction initiators, who are anonymous and cannot be trusted. Oracles mitigate this issue by attempting to provide a near-trustless way to relay data between a blockchain network and the real world (off-chain world) [8]. This enables the noticeable potential to enforce contractual relationships based on real-life data. Without the capability to interact with and process data from outside sources like sensors, API-ingestible data and countless other sources, decentralized applications on the Ethereum blockchain and any decentralized Web3 platform are extremely limited in capabilities, having only to rely on information and state available on the blockchain only.

However, the introduction of extrinsic data introduces a significant risk to the security and consensus model intrinsic to the design of the blockchain system [8,30]. The incoming data can be corrupted, manipulated, or tampered with and cause faulty execution of our smart contracts, which defines the 'Oracle problem', where, however secure a system is, corrupt input data always results in corrupt output (garbage in, garbage out). This is the Oracle problem: How can we ensure a secure and trustless bridge between the blockchain and the outside world? We summarize the classification of the Oracles in Table 1, inspired by [4], highlighting the characteristics of different oracles.

Table 1. Classification of Oracles by Data flow and Operating Paradigm.

	Variants		
Data Flow	*Inbound:* Oracles function only as a source of external data to the blockchain	*Outbound:* Oracles function as only an output bridge, taking data from the blockchain and relaying them to off-chain services	*Inbound–Outbound:* Oracles function as a two-way bridge of data between the blockchain and the outside world.
Operation Paradigm	*Request–Response:* Oracles receive requests from clients and fetch the data and return them as requested	*Immediate–Read:* The oracle acts as a lookup service, whereby it is queried and provides an instantaneous response	*Publish–Subscribe:* Oracles allow consumers subscription to their data output/input and push data whenever available.

3. Related Work

Attempts have been made to address the oracle problem by proposing various oracle designs that cater to different use cases, and we demonstrate some of them in this paper. We classify the literature into two types, *centralized* and *decentralized*.

3.1. Centralized

Provable (previously known as Oraclize) [33]: has been operational since 2015 as a safe way to fetch data from external Web APIs. Its main objective is to ensure verifiable and auditable off-chain data availability. It fetches data from these sources and stores them using authenticity proofs in a decentralized storage system like IPFS or SWARM. The model used by Provable is based on Trusted Execution Environments (TEEs) [34] and auditable virtual machines to build authenticity proofs. The major drawback of Provable is that it uses a centralized trust model representing a single point of failure.

TownCrier [35]: is similar to Provable, utilizes the concept of *TEEs*, and, namely, uses Intel's SGX (Software Guard Extensions) enclaves [36]. The main idea is that a central server houses the logic for receiving requests and fetching data inside a secure enclave designed to be completely isolated from the operating system or other applications that can interfere with the logic housed in this enclave. It provides attestations, authenticity proofs, and guarantees of integrity and confidentiality using private–public key pairs. Again, like its predecessor, the main drawback is the single-point-of-failure model, where trust is centralized.

Augur [37]: provides a low-cost oracle platform for prediction markets for online trading. It leverages the community's power to make informed decisions, where users on the platform who have reputation tokens can choose the outcomes of the prediction markets by staking their tokens on the actual observed outcome, and they receive settlement fees from the market based on their choices. Augur operates by opening up markets in four stages: (1) creating a market, (2) trading, (3) reporting, and (4) settling. In the first phase (creating a market), everyone on the platform is allowed to set an event and its time and select a designated reporter. However, that single reporter does not have the full power to determine the outcome, and the community has the ability to interfere with disputing the reporting and coming up with a different resolution. In the second stage (trading), Augur's trading contracts maintain an ordered book of every prediction market. In the third stage (reporting), reporting begins, and the oracle determines the actual outcome of the event. This reporting stage comprises seven stages and includes multiple rounds for possible disputes. Reporters who staked on the wrong outcome are penalized, and the final settlement happens in the fourth and final stage of the oracle phases (settling). The main concern with AUGUR is that it relies on crowd wisdom, which is not a reliable source of truth and is vulnerable to manipulation.

3.2. Decentralized

DECO (DECentralized Oracles) [38]: The protocol is designed to provide decentralized Oracle services for smart contracts on blockchain platforms. It aims to address trust and security issues in Oracle systems by leveraging trusted data sources, verifiable computation, reputation systems, and decentralized governance. DECO focuses on ensuring data integrity, offering a framework for secure and reliable data retrieval by smart contracts. By incorporating techniques like zero-knowledge proofs and reputation-based assessments, DECO aims to provide a decentralized Oracle solution that enhances the trustworthiness and security of external data integration in blockchain applications.

However, DECO has certain limitations that should be considered. One potential challenge is the reliance on trusted data sources, which may introduce a degree of centralization and contradict the goal of full decentralization. Establishing and maintaining trust in these entities can be a complex and ongoing process. Additionally, DECO's practical implementation and scalability may require further exploration and validation. As with any novel protocol, DECO's real-world performance and adoption need to be assessed to understand its effectiveness and viability in different use cases. These factors highlight areas for further research and development to optimize the effectiveness of the protocol and overcome potential limitations.

ASTRAEA [39]: A decentralized oracle takes advantage of human intelligence through a voting-based game to ensure the security of external data on the blockchain. It is a general-purpose oracle that runs on a public ledger and can be utilized on other platforms with similar smart contract characteristics. The system has three roles: submitters, voters, and certifiers, each with different levels of risk and reward. Submitters pay fees to submit Boolean propositions to the system, while voters and certifiers play a low-risk/low-reward and high-risk/high-reward game, respectively. The system has desirable properties, including a Nash equilibrium in which all players play honestly and a low probability that an adversary manipulates the voting outcome. However, some cons of the system include the possibility of collusion among voters and certifiers and the need for a large number of participants to ensure the security of the system.

dAPI (decentralized API) [40]: is a safe and economical way to provide smart contracts with a traditional API service in a decentralized manner. The goal of API3 is to create, administer and commercialize dAPIs on a scale [41]. In particular, API3 aims for decentralized APIs (dAPIs) by enabling first-party data providers to supply data directly to blockchain networks without middlemen. It creates a more trust-minimized setup by eliminating the need for third-party Oracle nodes. The consequence is that it places trust in API providers to supply accurate data, which could be a potential point of failure.

Chainlink [42]: is a widely recognized oracle network that provides decentralized oracle services for smart contracts on various blockchain platforms. It offers robust features to address the challenges associated with securely and reliably accessing external data. Chainlink's key features include a large network of oracle nodes operated by independent node operators, data authentication mechanisms, multiple aggregation methods, and reputation systems. Chainlink operates using a combination of on-chain smart contracts for aggregation and node selection and off-chain oracle nodes that retrieve the data and return them to the Chainlink contract, where they are aggregated and returned to the requesting contract. The reputation of Oracle nodes is evaluated and reported to an on-chain smart contract for transparency.

Nodes are chosen through a combination of bidding systems, where user-operated nodes bid on submitted jobs, and a combination of bidding fees and reputation determines the winning node. A smart contract on the blockchain performs this selection, and the committal of nodes provides a penalty fee that is returned after successfully completing the tasks. Nodes fulfill the task by fetching the data and submitting their responses to the network by calling the Chainlink contract. Results are verified using various cryptographic and authentication methods and returned to the user. An obvious problem with the Chainlink system is the cost of submitting the results to the blockchain, where each node

needs to call back the function to submit its results. When consumers request multiple data points from multiple sources, costs can add up. Also, no centralized ledger binds to the off-chain nodes, and their actions are not recorded.

4. System Design and Architecture

4.1. Overview and Design Philosophy

Veritas is a decentralized input oracle that mainly operates using the request–response paradigm and attempts to provide data securely from hardware and software data sources. The best security can be achieved by having decentralized nodes fetch data from multiple sources and then aggregate these results to guarantee the best outcome without any chance of a single point of failure or tampering that could skew the result by relying on only one source.

The 'Veritas' oracle network is a separate blockchain serving as a sidechain that is interoperable with Ethereum and uses the same exact consensus mechanism that Ethereum uses, meaning that the security protocols that enable robust and trustless operation on the main (layer-1) chain also apply to the side chain since it is a blockchain in its own right. A side chain operating in conjunction with a main chain is what is known as a layer-2 chain, as it represents a second layer in processing transactions. As mentioned in Section 2.2, Ethereum has recently undergone a transition to POS, which greatly improves efficiency and speed. Veritas is also POS-based, as POS guarantees faster transaction approval instead of the computationally intensive work required to approve a block using POW, and it is also better in terms of efficiency and power requirements. Each node in this side chain is a fully participating blockchain node with a node client communicating with it. Figure 1 illustrates the architecture and design of the system.

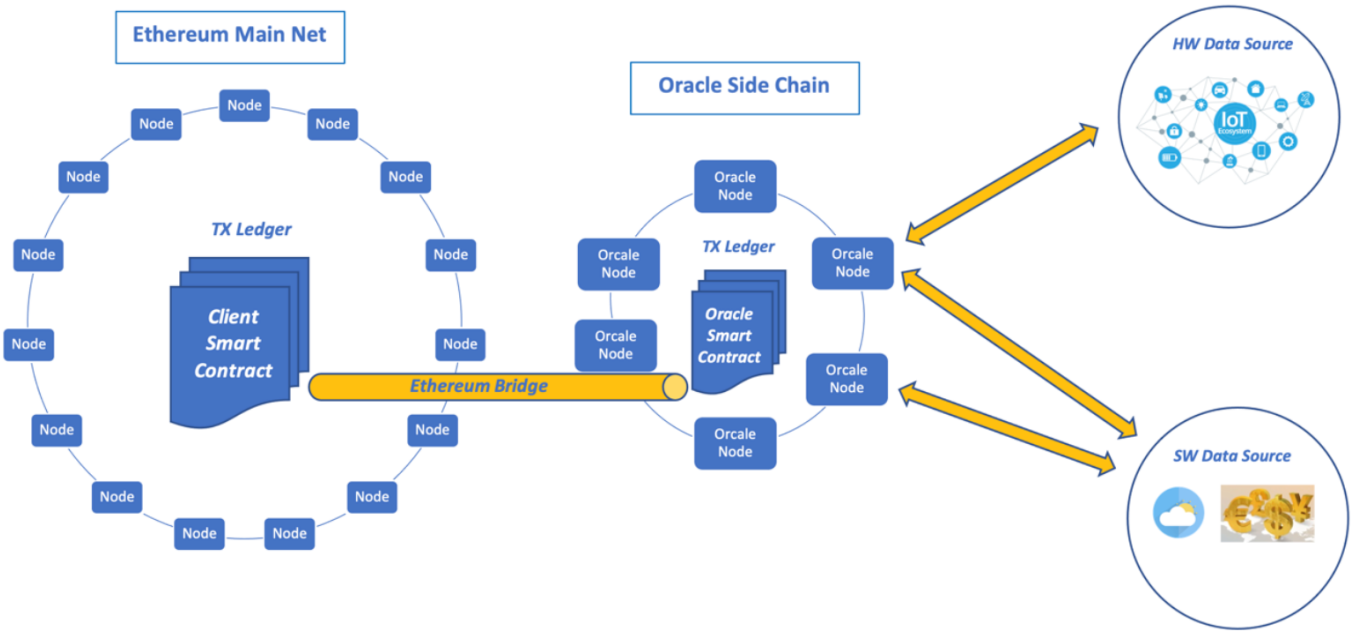

Figure 1. Veritas system Architecture.

The scope of this paper does not discuss the consensus mechanism of POS that we use in the Veritas network, as we assume it operates the same way as Ethereum does, where nodes that wish to verify transactions should stake a certain amount and are randomly selected to verify transactions as previously discussed.

The reasons behind selecting a side chain as the basis for the architecture are as follows:

1. A blockchain provides transparency as all requests to and from nodes are appended to the distributed ledgers for all parties to audit and verify instead of relying on off-chain nodes to fetch and aggregate results.

2. A side chain that operates outside the main chain (layer-2) achieves better scalability and throughput as the reduced number of nodes operating and the employment of a POS consensus guarantee more efficient operation.
3. A side chain also guarantees lower gas fees as the number of transactions is significantly lesser than in a congested main net. There is less contention for network resources to approve, verify, and append transactions to the ledger.
4. A blockchain allows the enforcement of node staking using the blockchain token 'VERT', which is discussed in further sections and is integral to the operation of the Oracle network.
5. Using a smart contract at the heart of that blockchain guarantees that all operational aspects are logged, verified, and secure, and the contract itself cannot be tampered with.

This is in contrast to other decentralized Oracle systems that aggregate either completely off-chain, which introduces security and trust concerns, or aggregate on the main chain itself (Ethereum, in this case), which is highly secure but extremely costly. Having a layer-2 blockchain (side chain) offers the possibility to have the best of both worlds: efficient costs and robust security.

4.2. Proposed System Architecture: 'Veritas'

The Veritas Oracle blockchain network consists of three main parts: *an Ethereum bridge*, *the Veritas blockchain with the smart contract (VeSC)*, and *Oracle nodes*.

4.2.1. The Ethereum Bridge

Blockchain bridges connect different blockchain networks to facilitate the transfer of assets or tokens, making blockchain interoperability more attainable. Bridges can be centralized, where a single node between two different networks relays transactions and assets between various networks, or they can be decentralized, where a group of nodes serves as the pathway between the networks and achieves a higher degree of security and transparency. The operation of the bridge itself is not within the scope of this paper, but for our design and operation, we assume it is a decentralized bridge. Through the bridge, requests are relayed from the main net and served by the Veritas oracle nodes, and the result is relayed back.

4.2.2. Smart Contract (VeSC)

The VeSC (short for Veritas Smart Contract) serves as the heart and brain of the Oracle network. It is a fully fledged smart contract containing all necessary functions to enroll oracles, accept and validate incoming jobs, assign jobs to nodes, aggregate results, relay them back to the requester, and punish bad actors. When the network starts, the first node publishes this contract, which acts as the orchestrator of operations for the network moving forward.

4.2.3. Oracle Node

Oracle nodes operating in the Veritas Oracle network are full blockchain nodes with clients running together. These nodes have a local copy of the entire ledger; they can validate transactions, issue transactions, implement Ethereum's node specifications, and expose a JSON-RPC. The only functionality missing from the Oracle nodes is deploying smart contracts. Nodes implementing the Ethereum specification expose a JSON-RPC interface, allowing for external software (or clients) to connect to them and perform various operations and functions like issuing transactions, interacting with smart contracts, and enabling the web3 or the DApp ecosystem. A node in the Veritas network has an on-chain component (the full node) and an off-chain component (the client) that communicates with the on-chain component through the JSON-RPC interface exposed by the entire node. The client receives communications from the blockchain through transaction receipts (events) and can talk to the blockchain by issuing transactions through the entire node it is connected to. The node and client form what we call 'the Oracle node' throughout this paper.

4.3. System Operations

4.3.1. Initiating Job Requests

When a client smart contract on the Ethereum main net requires external data, it creates a transaction and sends it to the VeSC through the Ethereum bridge. There are two types of jobs that Veritas can serve: *API Request* jobs and *Real-Time (RT)* jobs.

Each requires some basic attributes necessary for either of them, like (but not limited to) the following:

- $minNoOracles[integer]$,
- $maxNoOracles[integer] >= minNoOracles$,
- $sources[arrayofurlstring]$,

Where *minNoOracles* field denotes the minimum number of oracles required to respond for a result to be valid, and *maxNoOracles* is the maximum number of oracles required to be assigned the job. The desired number the client wishes to have is the result aggregated from an *sources* array is a list of URL strings from which the client wishes to fetch the data.

Data APIs sometimes have keys or authentication mechanisms that need to be added to the request, but since they are sensitive and cannot be exposed, they need to be encrypted. We do not discuss them in the scope of this paper and assume the URL strings are sufficient in themselves to obtain the necessary data.

The design aims to minimize the number of calls to the main net as much as possible because each call involving the main net incurs a significant amount of gas (as previously discussed). To that end, we decided on a scheme to avoid multiple round-trip calls to estimate the fee: the consumed (client requesting data) submits a job by calling the VeSC, and an estimate id calculated accordingly. The fees are standardized according to the network load and posted in a public setting (a website). The cost is a function of the number of requested oracle nodes (max and min) and the number of URLs requested, where the total number of external calls is denoted by N_{Call} as seen in Equation (11). N_{Oracle} is the total number of oracles who make an external call and provide an answer, and N_{Source} is the total number of sources queried for the results.

$$N_{Call} = N_{Oracle} * N_{Source} \tag{3}$$

The client provides the necessary funds to perform the job with the job request, nullifying the need for a round trip to estimate gas fees. After the VeSC receives the job request, it is validated, and on success, it is added to the job queue assigned to nodes. After they respond, the results are aggregated and sent back to the requesting client by issuing a transaction to a callback function that the client provides in their smart contract.

Clients pay only for the number of responses that are collected from nodes. While a client pays for the maximum number specified in the request, if a subset of that number actually replies, they are refunded for the amount that does not reply, ensuring a fair balance of services vs. cost.

4.3.2. Node Enrollment

One of the most problematic issues with decentralized networks is 'Sybil attacks'. Sybil attacks are when participants in a network masquerade as different identities but are controlled by a single entity. A situation such as this can cause significant problems with the decentralized consensus mechanisms used in blockchains, be they using POS or POW, because, with enough nodes in the network, they can game the fetched results or even tamper with transaction validation. The solution is to make the participation of nodes in the data-fetching process expensive.

Therefore, the design of the system demands that before a node is permitted to join the pool of nodes that are assigned jobs, they stake an amount of the Veritas token 'VERT'. This amount is unrelated to whether the full node itself is staked and validating transactions on the Veritas blockchain; this is a separate stake sent to the VeSC and stored on the contract for

the node's participation in the Oracle network activities. This staked amount is collateral against any possible fraudulent behavior, where bad actors are penalized by claiming that part of their staked assets will never be returned.

This incentivizes good behavior and encourages nodes to act in good faith and accomplish their assigned jobs without any interference or gaming of results.

When enrolling, nodes are encouraged to provide additional KYN (Know your node) data. This may include data like passport or national identity numbers and images to confirm identity. These data are not stored on the blockchain for security purposes but on a separate Interplanetary File System (IPFS). Obliging nodes reduces their stake requirements, contributing to a more transparent operational paradigm for nodes that choose not to operate anonymously. Verifying the provided information on nodes is outside the scope of this paper but can be accomplished by contacting third parties and verifying the sanity of the data.

The longer a node operates honestly in the network, the more significant its reward becomes over time. However, unlike other decentralized oracle systems, they are not more likely to be picked to fulfill a job because of the length of time they are enrolled in the network. The only factor affecting their higher probability is their reputation, as discussed in further sections.

4.3.3. Node Selection

Node selection is a critical design aspect of our system, and a challenging one. In order to 'level the playing field' and not compromise on security, instead of bidding protocols (like those introduced in Chainlink [42]), the choice here is based on a multitude of factors, namely outlined below.

Random Seed

As discussed before, achieving randomness is very difficult in a blockchain context because all data are available to all participating nodes in the ledger. There cannot be a random function that executes differently on every participating node, resulting in an incoherent state; therefore, the Ethereum network does not define an inherent way to generate a random number. However, we can rely on a pseudo-random number, which is the latest transaction hash and is unpredictable for participating nodes.

Node Reputation (Based on Historical Performance)

Node Reputation refers to a score maintained on the VeSC that denotes the rating of a node. It is based on its historical performance. A node's initial score (out of 10) is 10. Every time a node is assigned a job, and its results are included in the final result, it maintains that score. If a node fails to report a result that is acceptable and included in the final result, it loses 0.5 points. It regains 0.5 points after achieving two successful jobs where the result is accepted in the average result. The nodes are classified into 'Integrity pools' as depicted in Table 2. Any node below a score of five is automatically removed from the node pool, and whatever stake is left is refunded.

Table 2. Node Integrity scores in Integrity pools.

Reputation	Integrity Score
High	$8 \leq$ Integrity Score ≤ 10
Medium	$6 \leq$ Integrity Score < 8
Low	$5 \leq$ Integrity Score < 6

Load Balancing

Nodes must have a fair chance of having an equal opportunity to be assigned jobs, so the design does not favor long-living nodes in the pool to prevent potential gaming of the results by nodes assigned many jobs. This makes it difficult for nodes on the network

to use this advantage for a long time to collectively manipulate the results, assuming that they attempt a Sybil attack. This aims to eliminate any advantage the old nodes may have. The fact that they may be more likely to be chosen because they have been on the network for longer constitutes security risk. Therefore, an algorithm was designed for all of these required properties. This algorithm is known as the *Veritas Selection Algorithm (VSA)*. The following explains VSA for choosing nodes to complete a given job.

Reputation Factor

There are some necessary steps to compute the reputation factor as follows:

1. The client specifies the number of nodes required for a job.
2. A client can select one node for a simple decentralized job.
3. The base case for a decentralized job requires two oracles.
4. In the case of one Oracle, a high-integrity node is assigned.
5. In the case of two Oracles, a high-integrity and a medium-integrity oracle is assigned.
6. In the case of three Oracles, an additional low-integrity Oracle is assigned.
7. The client needs to pay an extra fee if they choose to have lower-reputation nodes replaced with higher-reputation ones.
8. For each successive oracle, a high one is added, then a medium one, and then a low one unless otherwise specified by the client.

Randomness and Load Balancing Factors

The algorithm applied to each reputation pool to achieve sufficient randomness and load balancing is explained in Algorithm 1.

Algorithm 1 tries to achieve load balancing and introduce a randomness element so that, over time, all nodes achieve equilibrium in the number of assigned jobs. This algorithm can be summarized as follows:

1. Offer A, which is the indexed array of nodes where $A[i]$ is the number of jobs that were assigned to the node at index i.
2. Offer N_{Oracle}, which is the length of the array and the total number of nodes available in the pool.
3. Calculate M, where M represents the target number of jobs that all nodes in the pool should have at least been assigned as shown in Equation (4).

$$M = \lceil \frac{1}{N_{Oracle}} \sum_{i=0}^{N_{Oracle}-1} A[i] \rceil \tag{4}$$

   Also, save value S_j mentioned in Equation (5) as this value is used to recalculate the median as jobs are assigned without having to reiterate over the array and exhaust resources.

$$S_j = \sum_{i=0}^{N_{Oracle}-1} A[i] \tag{5}$$

4. For an incoming job, fetch the last number in the TX hash of last transaction T.
5. Calculate R as shown in Equation (6).

$$R = T \bmod N_{Oracle} \tag{6}$$

6. Check whether $A[R]$ satisfies the 'assignment criteria', where the criteria should be that the node has been assigned jobs $< M$
7. If the node has been assigned several jobs less than the median, it is assigned the job, and $A[R]$ is incremented.
8. If not, check whether R is even or odd; if it is even, then start iterating from $i = R$ to $i = 0$ in search of a node that fulfills the criteria. If R is odd, iterate from $i = R$ to $i = N_{Oracle} - 1$.

9. If a node is found that satisfies the criteria, it is assigned the job. If the loop is concluded before a node is found, that means all nodes have been assigned at least the median amount of jobs, and the job can then be assigned to the node at $j = R$, and the index is incremented.
10. M is recalculated by substituting the previously computed S_j, where

$$M = \frac{S_{j+1}}{N_{Oracle}} \tag{7}$$

Algorithm 1 Choose Oracle Node

Require: Indexed array A of Oracle nodes, where $A[i]$ is the total number of jobs performed by Oracle at index i.
Ensure: Index of chosen Oracle node to perform a given job

Obtain T, where T is the last number from the previous TX hash
Calculate R, where R is the result of T mod N_{Oracle}, where N_{Oracle} is the number of oracle nodes in array A
Calculate M, where M is the median number of jobs performed by oracles in the pool calculated by $M = \left\lceil \frac{\sum_{i=0}^{N_{Oracle}-1} A[i]}{N_{Oracle}} \right\rceil$
Save the value for $\sum_{i=0}^{N_{Oracle}-1} A[i]$ as S_j that represents the total sum of jobs
if $A[R] < M$ **then**
  $A[R] \leftarrow A[R] + 1$
  return R
end if
if R is even **then**
  for $j = R$ to 0 **do**
    if $A[j] < M$ **then**
      $A[j] \leftarrow A[j] + 1$
      return j
    end if
  end for
  $A[R] \leftarrow A[R] + 1$
  return R ▷ If loop finished and did not find a suitable oracle
else
  for $j = R$ to $N_{Oracle} - 1$ **do**
    if $A[j] < M$ **then**
      $A[j] \leftarrow A[j] + 1$
      return j
    end if
  end for
  $A[R] \leftarrow A[R] + 1$
  return R ▷ If loop finished and did not find a suitable oracle
end if
Re-evaluate $M = \frac{S_{j+1}}{N_{Oracle}}$

4.3.4. Request Fulfillment for API Jobs

Figure 2 illustrates the API job lifecycle, which starts with the client initiation job request to VeSC. After the job's validation, oracles are selected according to Algorithm 1, and then an event is emitted with the chosen oracle addresses and the job details. The Oracle clients attached to the on-chain counterparts pick up those events by listening in on those emitted events, and chosen oracles check the job for the URL(s) in the job details and fetch the data accordingly.

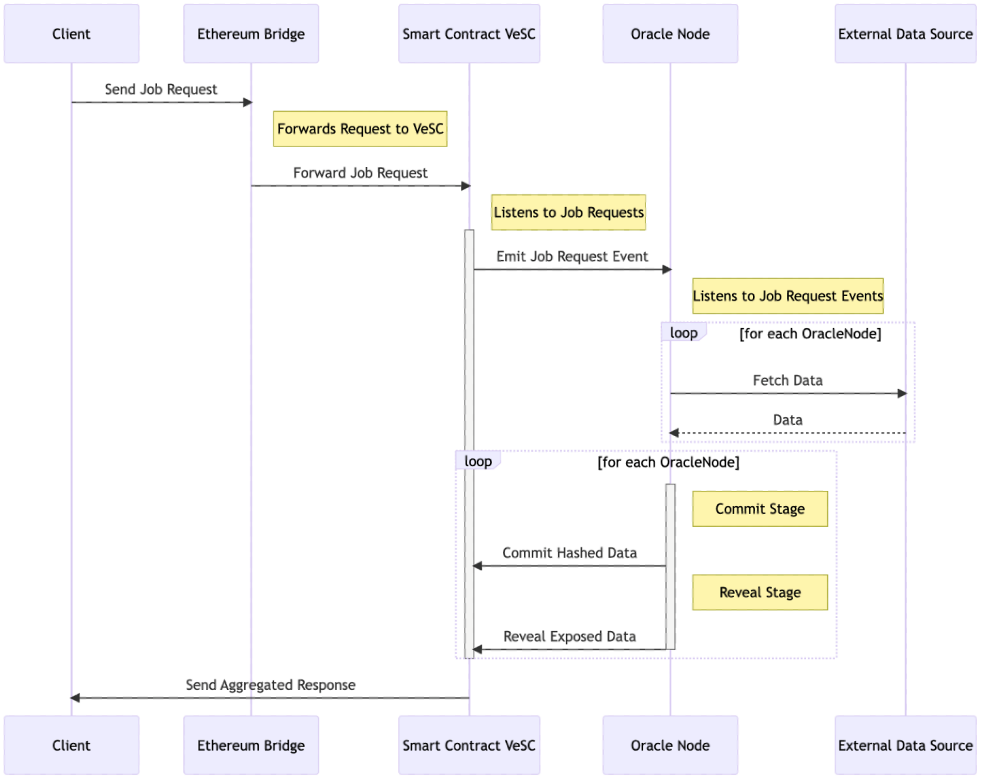

Figure 2. API Job lifecycle sequence diagram.

A method of dishonest behavior among such decentralized systems is 'freeloading', whereby oracles can copy each other's answers. We combat this by forcing nodes to send their results in two steps:

- **Step 1:** Committing to a result, where nodes send a hashed result that hashes the result with their node address on the blockchain and a random number generated off-chain. This result is sent and stored on the blockchain.
- **Step 2:** Revealing the result by sending it along with the random number previously used for hashing. The smart contract then takes this result, hashes it using the incoming random number and the node address, and compares it to the already saved hash. If they match, the node is not copied or forged; otherwise, it is penalized.

This is known as a *'commit–reveal'* scheme and is used in other decentralized data-fetching mechanisms to combat freeloading. The data are then cross-checked and aggregated to reveal a final result. The scope of this paper does not include improvements or analysis to truth-finding mechanisms or algorithms to detect outliers. Veritas currently proposes to use simple aggregation and standard deviation to pinpoint outliers and penalize nodes that provide such results. A minimum number of nodes (as specified by the job parameters) must respond with results to fulfill the job. Nodes that do not respond are penalized. The response is then sent to the requesting contract on the main network via the Ethereum bridge.

The gas cost for fetching data in the API-request mode using Veritas, Veritas Gas Cost ($VeGC_{APIReq}$) can be deduced from Equation (8). Ethereum Transaction Gas Cost ($EthGasCost_{Trans}$) is the gas cost required to perform a transaction back to the Ethereum network to ship the aggregated data back to the Ethereum main net. Node Transaction Gas ($Gas_{NodeTrans}$) is the gas required while submitting one data point from an oracle node to the VeSC and N_{Res} is the total number of responses from the nodes to the VeSC. Finally,

the $Cost_{VGas}$ is the Veritas gas cost, which, as previously discussed, is substantially lower than $EthGasCost_{Trans}$ since Veritas is a smaller side chain.

$$VeGC_{APIReq} = EthGasCost_{Trans} + Cost_{VGas}(Gas_{NodeTrans} * N_{Res}) \quad (8)$$

4.3.5. Request Fulfilment for Real-Time Jobs

A common kind of a data-fetching job requires a continuous stream of real-time data. An example would be currency exchange data like the price of ETH/USD or USD/EUR, or it could be for fetching IoT data from sensors or weather data from publicly exposed APIs. Applications that demand this data streaming mode vary, but maintaining secure and auditable real-time data remains essential. This can be for medical, logistics, military, or any information-sensitive domain where the integrity of the data is paramount. However, these kinds of jobs could be costly due to the continuous transactions to supply data to the main blockchains, where each transaction requires a substantial cost associated with it, so we devised a method (Figure 3) for making the process more efficient as follows:

1. A client (consumer) requests a real-time job for N nodes from a data source(s).
2. They specify a 'deviation' amount Dev, that is, the % where if the new aggregated data point value deviates from the last reported data point value, it is reported back to the client. Otherwise, if the deviation is below that value, the data point is not reported back.
3. A boolean flag R, where if true, the nodes report every data point they fetch regardless of the deviation back to the VeSC, and after aggregation on the VeSC, the VeSC determines whether the aggregated data point deviated more than the value Dev and decides whether or not to send the data point back to the main Ethereum network.
4. Duration of job D, which denotes how long the job should run for.
5. Frequency F, which denotes how often nodes should check the external sources for data.
6. Optional Heartbeat duration H, which denotes the maximum time between each data point (regardless of the deviation) and ensures that nodes are still alive and serving the request properly.

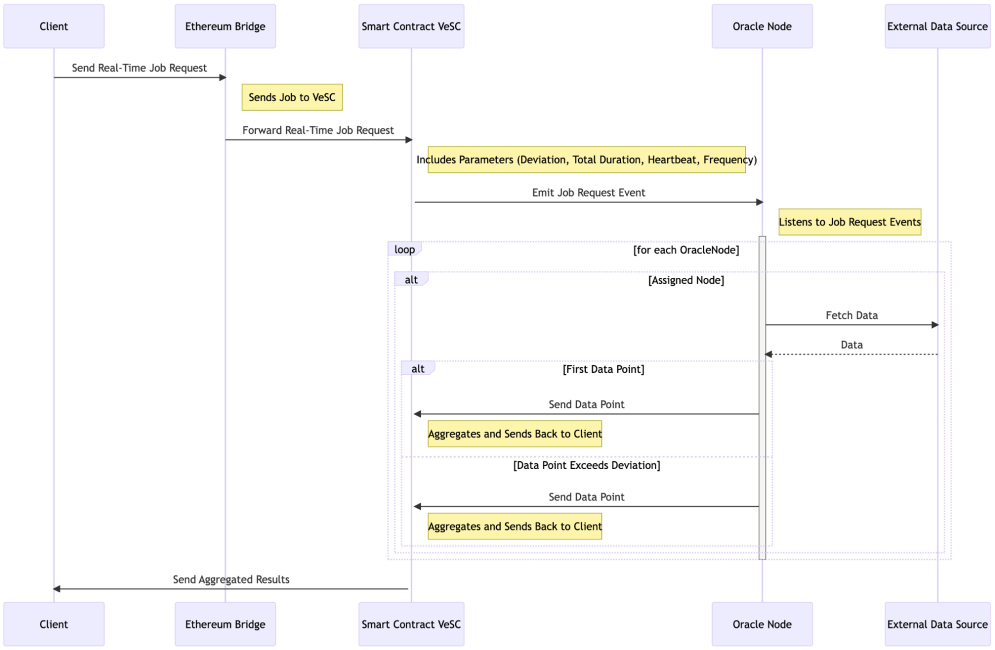

Figure 3. Real-time Job lifecycle sequence diagram.

The total number of calls made back to the main Ethereum network with valid data points can be modeled using Equation (9). F is the frequency of polling for data from exter-

nal sources in requests per second (req/s). V denotes the percentage of valid aggregated data points that deviated from the configured value (expressed as a decimal). D is the duration of the job in seconds.

$$N_{Req} = 1 + F * V * D \quad (9)$$

After that, the total gas cost for a real-time job on Veritas ($VeGC_{RTReq}$) is calculated using Equation (10).

$$VeGC_{RTReq} = EthGasCost_{Trans} * (1 + F * V * D) + Cost_{VGas} * (Gas_{NodeTrans} * N_{Res}) \quad (10)$$

The number of nodes is irrelevant since the data are aggregated from the N_{Oracle} nodes and reported back to the main net only when the aggregated value exceeds the deviation configured for the job.

It should be noted that if flag R is false, it is up to the individual node to determine whether the data point that was fetched deviated from deviation amount Dev and then sent back to the VeSC. In the case of R, this was true, which provides extra transparency where all data points fetched are reported to the side chain, and the process of aggregation and determination is transparent. This implies extra costs, as each node has to initiate a call every time it polls for new data. Another point that can be configured, which we do discuss in detail here, is whether each node should aggregate data fetched from multiple sources off the side chain or on the side chain. Aggregating on the side chain means that each node initiates

$$N_{CalltoVeSC} = N_{Source} \quad (11)$$

5. Experimental Evaluation

This section discusses our evaluation of the network design proposed in Section 4 and how it compares to 'Chainlink' [42]. We chose 'Chainlink' because it is the industry standard for decentralized oracles for Web3 application development. As mentioned above, our design relies on the premise that our network is a side chain. This blockchain operates with the same consensus rules as Ethereum, where nodes validate transactions on the public ledger. To that end, we simulated the sidechain locally using Ganache [43]. This simulation tool can create an entire blockchain network with any number of simulated nodes with desired public and private addresses. The simulation for Veritas is mainly a POC to verify our proposed design of harnessing the concept of layer-2 blockchains (sidechains) and employing it to serve the crucial function of providing the mainstream blockchains with data in a cheap, decentralized, secure, and transparent manner.

With the lack of decentralized Oracle peer-reviewed academic research and implemented POCs with comparisons to Chainlink, we have no specific metrics to compare the performance of Veritas against, especially since we are not considering the block mining time or efficient gas consumption. We are contributing a more efficient and scalable design for fetching data into mainstream blockchains. That is why we also roughly simulated the Chainlink operator node and data fetching functionality to assert our understanding of Chainlink operation in light of the absence of academic papers on the details of its operation.

5.1. Experimental Setup

Next, we discuss the Veritas and Chainlink setup. All experiments were implemented on a Macbook Pro 2018 with a 2.2 GHz 6-core Intel Core i7 with 16 GB of RAM.

5.1.1. Veritas Setup

In order to validate the Veritas design, we implemented a complete smart contract (VeSC) to run on a local blockchain that simulates the sidechain concerning Figure 4 using the truffle development suite to develop the contract using the Solidity programming language, the industry-standard language to write smart contracts to be run on the Ethereum

network and executed on blockchain nodes' EVMs. We run a local blockchain through Ganache with n nodes. The smart contract was deployed on the sidechain using the address of the first local node. A front-end webpage triggered the sending of an incoming job to the sidechain using the address of the second local node. We also used a node.js script that simulates a single Oracle node and allocates an address from the available local sidechain nodes. We also used another node.js script that instantiated $n-2$ nodes by running the script above $n-2$ times while allocating different ports and a different blockchain node address. Finally, an express.js server ran locally to simulate a data source that provided mock data through an API call.

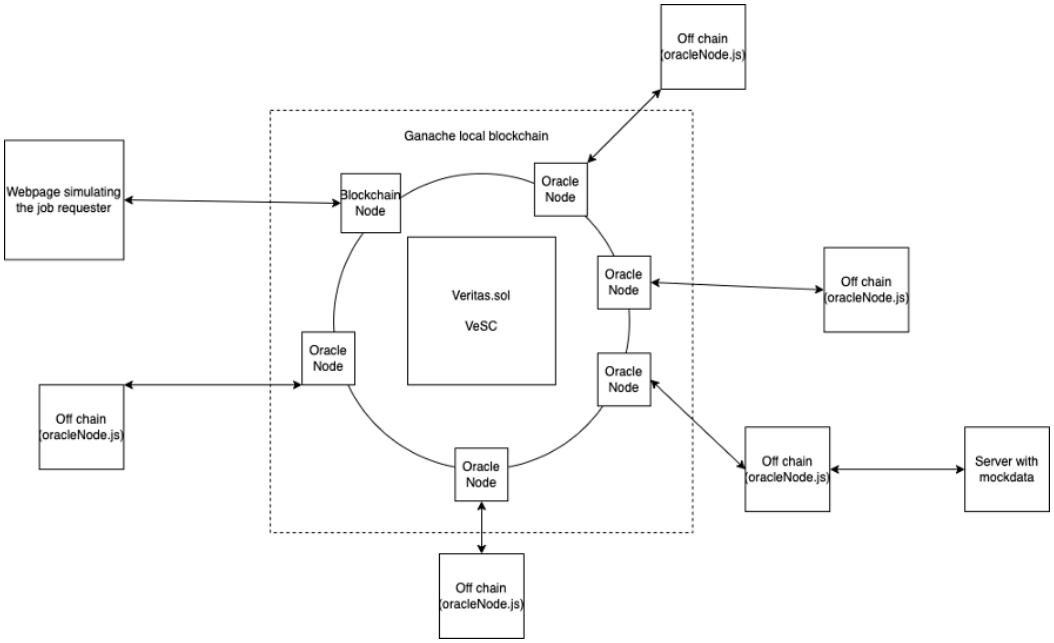

Figure 4. Veritas Experimental Setup.

There were some necessary assumptions to perform the simulation. First, nodes had enrolled and deposited the required cryptocurrency tokens (VERT). In addition, the Ethereum bridge exists, and the job call arriveed through that bridge, simulated through a call from an existing node on the blockchain. We also assumed that the blockchain automatically mines and approves the transactions and submits blocks directly to the blockchain and that block verification and transaction appending are outside the scope of this simulation. Ganache automatically handled this. Finally, sending the result back through the bridge was outside the scope of this simulation.

5.1.2. Chainlink Setup

We set up a local Chainlink operator node to compare gas consumption results. An operator node is an Oracle node whose users can set it locally on their machine and start serving jobs. We aim to validate our understanding of how Chainlink operates when serving requests from the blockchain.

This setup also uses Ganache, but in this instance, it simulates the Ethereum blockchain. For a request to be served by Chainlink, a consumer (client requesting data) includes a Chainlink library in their code and initiates a request to fetch some data from a URL. This initiates a call to another smart contract, the smart contract of the Oracle node assigned to the job. Every oracle has its smart contract deployed to the Ethereum network that is contacted when the bidding process ends and the address/node is assigned the job. We run a local blockchain through Ganache with n nodes. Along that, a smart contract, `Consumer.sol`, is deployed on the local blockchain simulating the smart contract by the consumer requesting the data from an oracle. We use Docker and Kubernetes [44] to orchestrate a number of containers running as follows:

- The Nginx server to host the oracle operator interface webpage;
- The Chainlink off-chain component that is allocated an address from the Ganache network;
- A PostgreSQL database to save the fetched results (optional).

Additionally, we have a node.js script that triggers the request by using Truffle CLI and finally another node.js script that monitors the transactions by picking up on the emitted events. For the Chainlink simulation, we assume that the bidding process is concluded and that the node to serve the request is chosen. We also assume that the simulation considers a simple request to serve data from a single node from two different URLs. Finally, we assume that the simulation considers Ganache a simulator of the main Ethereum blockchain.

5.2. Experimental Results

In this section, we present two experiments: *Experiment 1* for simulating the API Request job (Section 5.2.1) and *Experiment 2* for simulating the real-time job (Section 5.2.2).

5.2.1. Experiment 1: Simulation of API Request Job

For the Veritas simulation, we deploy the VeSC contract on the blockchain. We then request a job (simple API fetching job from 3 nodes and 2 data sources). The VeSC validates the job, selects the nodes from the pool, and then emits an event containing the addresses of the selected nodes for fulfilling the job along with job parameters.

All nodes receive the event through the JSON-RPC interface, and those nodes that find their address in the job assignment start performing the data-fetching operation on the mock server. Nodes then retrieve the data and perform the 'commit' phase, where they hash the plain text result with their address and a random seed and send it to the smart contract (VeSC), which then stores the hashed result. The nodes then send the revealed result along with the random seed used. Afterward, the VeSC takes the revealed plain text result, hashes it with the node's address and random seed, and matches that with the stored result. If it is the same, then the node has indeed committed a result and not freeloaded from other nodes, and the result is considered in the aggregation. Finally, once all (or the minimum number of nodes for the job) nodes answer, the VeSC aggregates the result.

For Chainlink simulation, through script `req-eth-price.js`, we trigger a call to the `Consumer.sol` smart contract that simulates a consumer attempting to fetch data from Chainlink. The contract has already defined the oracle to serve the request, and it calls the Chainlink library to package a job request, which emits an event with the job parameters. Moreover, the node picks up the event, fetches the data from the URL, and then calls back the smart contract using a predefined callback function in the job.

We then repeat the experiment for an increasing number of nodes and compare the overall gas consumption between Veritas and Chainlink.

Results for Experiment 1: Simulation of API Request Jobs Using a Single Oracle Node

This experiment aims to determine the gas consumption for an API request job using a single Oracle node in Veritas and to compare it with Chainlink. As shown in Table **??**, gas consumption is directly related to how the smart contract is written and implemented and reflects the amount of computation that the execution of the target function requires. To this end, we present a comparison in the amount of gas consumed as a basis for the next section, where we can estimate the cost savings for fetching data using Veritas.

The gas consumed reported in these findings represents the computation required to perform the operations listed. Each operation performed on an EVM requires an amount of gas (as described in Section 2.2.4). Gas price differs, as explained later in the discussion and interpretation of the results. It should be noted that the Chainlink simulation we achieved is bare bones. It does not simulate the commit–reveal process or consider the node selection, both of which consume additional gas. The cost difference between Veritas and Chainlink could be demonstrated through Figure 5. The cost of fetching a single data

point using Veritas can be seen in Equation (8), while for Chainlink, it can be determined by Equation (12).

$$ChainlinkGC_{APIReq} = EthGasCost_{Trans} * N_{Res} \quad (12)$$

Table 3. Comparison between Veritas and Chainlink with regard to gas cost of operations. We note that N/A for Chainlink denotes that we did not replicate the commit phase as we have no information on how Chainlink performs this operation.

	Veritas		Chainlink	
	Operations Executed on VeSC	**Gas Consumed**	**Operations Performed in Consumer.sol 3**	**Gas Consumed**
Submit a Job	1. Validate Request 2. Select Nodes 3. Create Job request 4. Emit event	489,148	1. Create Job request 2. Emit event	139,529
Oracle response with a commit	Save the hashed result	44,562	N/A	N/A
Oracle response with plain text	1. Hash the incoming result with random seed and address 2. Verify the hash equals saved hash 3. Aggregate the results and emitting a 'completed' event if a minimum number of nodes respond	83,336	1. Save result 2. Emit 'completed' event	54,423

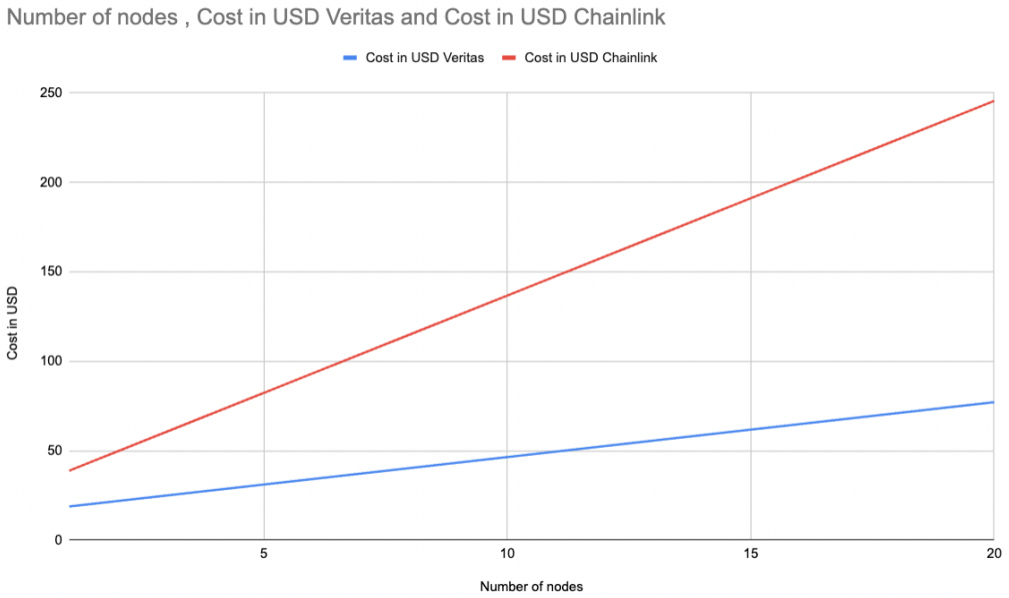

Figure 5. Veritas and Chainlink Gas Cost for API Request–Response Job.

It can be demonstrated through Figure 5 and Equations (8) and (12) that the cost is substantially lower for Veritas given that the high gas price for transactions to Ethereum (Gas_{Trans}) is constant, while for Chainlink it is multiplied by the number of responses from the nodes (N_{Res}). This cost efficiency, as expected, increases with the number of nodes providing results, as is the case for a decentralized Oracle system. This is because Chainlink performs the aggregation of different nodes on the Ethereum chain itself.

5.2.2. Experiment 2: Simulation of Real-Time Jobs

Simulation of real-time jobs is very similar to that of API Request Jobs because they use the same setup and configuration. The difference is how the responses are sent from the Oracle nodes and how frequently we send the result back to the main Ethereum network. In this experiment, we assume that the parameters for the job request data are shown in Table 4.

Table 4. The parameters of job requests.

Parameter	Value
Number of nodes N_{Oracle}	3
Deviation *Dev*	5%
Frequency *F*	1 min
Duration *D*	15 min
Boolean flag *R*	false

As we present in Table 4, we assume that each node fetches data from 1 external source, and we assume that the value changes by 5% (regardless of higher or lower) every 3 min. Every 3 min, we obtain a valid data point from the node, which is then aggregated on the VeSC, where it determines whether it exceeds threshold *Dev* and sends back to the Ethereum main net should the aggregated value exceed *Dev*. Finally, we consider that value *V* represents the percentage of valid data points to be sent back for the duration of job *D*.

In this scenario, we received 15 responses from our oracle nodes (each of the three nodes sent data back to VeSC 5 times, once every 3 min), and each transaction to send data from the node to VeSC cost 64,453 gas units. This is because the data deviated once every 3 min by 5%; the rest of the time, the data were assumed to be the same or deviating by lesser threshold *Dev*.

When the contract receives the three responses, they are aggregated and sent back to the requesting contract on the Ethereum main net if the aggregated amount deviates by more than *Dev* from the last data point. The total number of requests to the Ethereum main net for Veritas can be determined by Equation (9) and the total cost can be determined by Equation (10). In comparison to Chainlink, which aggregates responses on the main Ethereum chain, the number of requests returned assuming the same parameters would be computed using Equation (12).

$$N_{Req_{Chainlink}} = F * D * N_{Oracle} \tag{13}$$

The total gas cost for Chainlink can be calculated using Equation (14).

$$ChainlinkGC = EthGasCost_{Trans} * F * D * N_{Oracle} \tag{14}$$

As can be seen by comparing Equation (9) to Equation (13), for Veritas, the number of requests depends on *V*, which limits the number of transactions sent back to the Ethereum main net. Moreover, aggregation is performed on the Veritas side chain, meaning that the number of requests is not dependent on the number of nodes. In contrast, Chainlink aggregates on the Ethereum blockchain, meaning that every node's data point is sent back to Ethereum through a separate transaction, substantially increasing cost.

5.2.3. Experiment 3: Veritas Node Selection Validation

We performed two experiments to validate our choice of node selection algorithm. Both experiments were implemented using a simple node.js script to model the algorithms. Next, we present the following experiments.

Validation of Removal of Bidding Process from Node Selection

An experiment is conducted to validate that the bidding process performed by Chainlink potentially jeopardizes the selection of nodes because it may offer malicious nodes an advantage to underbid the competition. For this experiment, we have a node.js script that simulates several nodes, and the script runs two scenarios:

1. **Scenario 1: representing Chainlink's bidding** where we simulate the bidding performed by assuming that 20% of the nodes are malicious and that half of those nodes

are prepared to severely underbid to increase the chances of them being chosen to accomplish a job. Then, we randomly choose three nodes from the pool. We repeat this operation 100 times and check how many malicious nodes are selected in each iteration.

2. **Scenario 2: representing Vertias's approach** is given the same parameters as Scenario 1 (20% malicious nodes), but we ignore the bidding process and randomly choose the nodes from the pool. We repeat this operation 100 times and check how many malicious nodes are selected in each iteration.

We repeat each scenario 10 times and record the results that are shown in Figure 6.

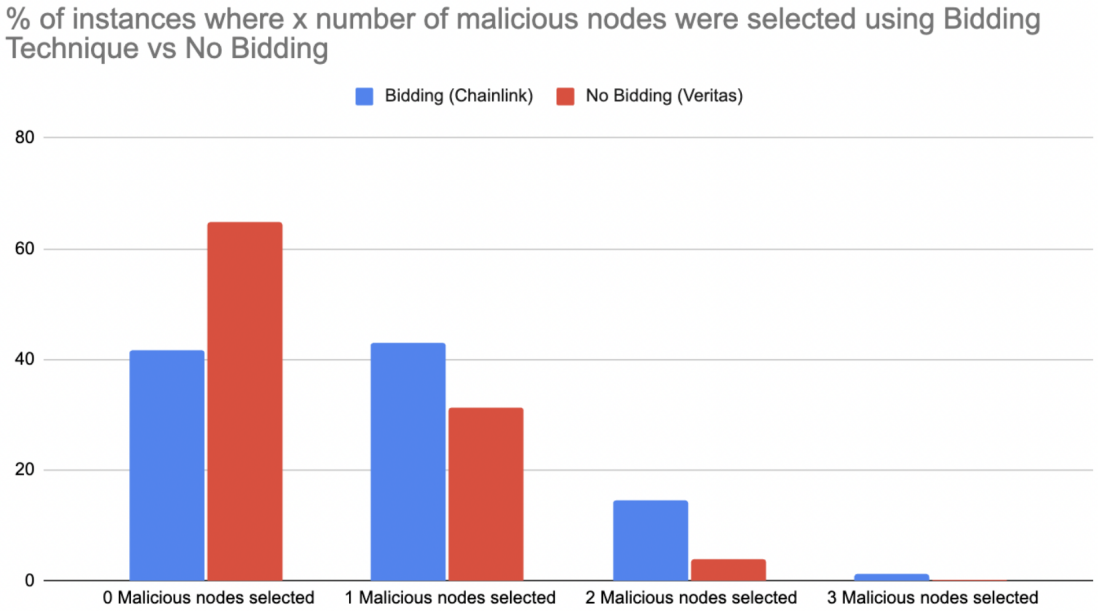

Figure 6. Simulation to demonstrate the effect of allowing nodes to bid for jobs as is performed in Chainlink vs. random selection in Veritas

As can be seen, the bidding process allows for malicious nodes to underbid and increases the likelihood that they are selected to perform jobs, resulting in questionable information. On the contrary, the likelihood is lower for Veritas, where there is no such process.

Validation of Node Selection Prioritizing Load Balancing and Randomized Selection

An experiment is conducted to validate our choice of load balancing and randomize the selection for every reputation pool. In this experiment, we demonstrate the operation of the node selection algorithm mentioned in Section 4.3.3. This experiment has four variants:

- **Variant 1:** An initial pool of 15 nodes with 0 previously assigned jobs; 500 incoming jobs are simulated and nodes are chosen.
- **Variant 2:** An initial pool of 15 nodes with a random number of jobs previously assigned; 500 new incoming jobs are simulated and nodes are chosen.
  As can be seen from the variants in Figures 7 and 8, the distribution of jobs using the load balancing technique employed for the Veritas system is flat compared to the random assignment technique.
- **Variant 3:** An initial pool of 15 nodes is simulated with 0 previously assigned jobs and 500 incoming jobs, and an additional new node joins the pool every 12 jobs.
- **Variant 4:** An initial pool of 15 nodes is simulated with a random number of previously assigned jobs and 500 new incoming jobs, and an additional new node joins the pool for every 12 jobs.

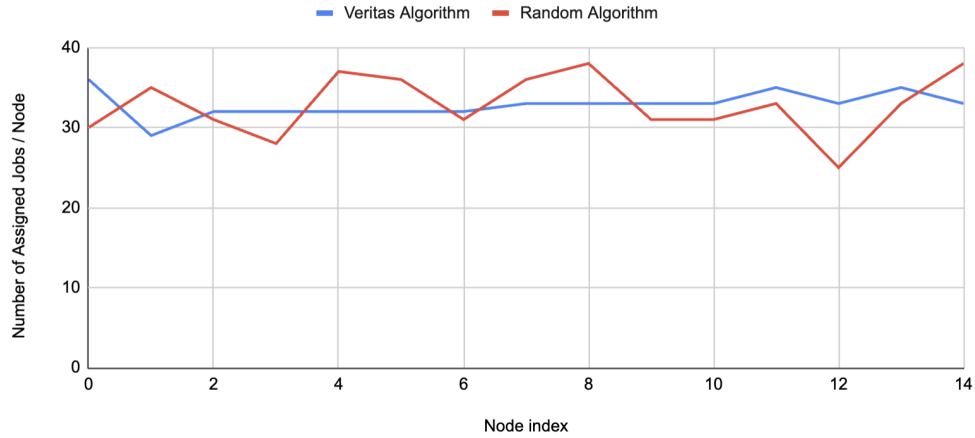

Figure 7. Node Selection Algorithm—Variant 1.

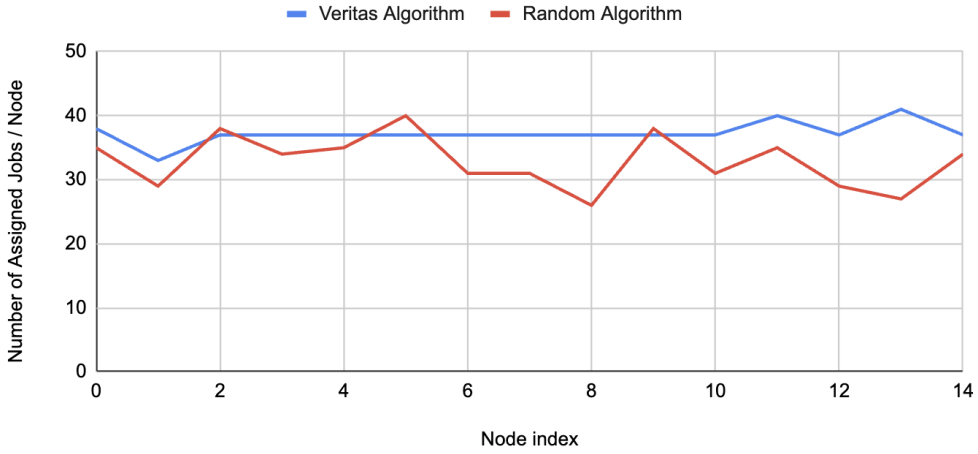

Figure 8. Node Selection Algorithm—Variant 2.

As can be seen from the variants in Figures 9 and 10, the Veritas selection algorithm considers new joining nodes and assigns them jobs to make sure there is no huge gap in job distribution and offer recent nodes a chance to gain reputation and not cause a monopoly for older nodes in the system.

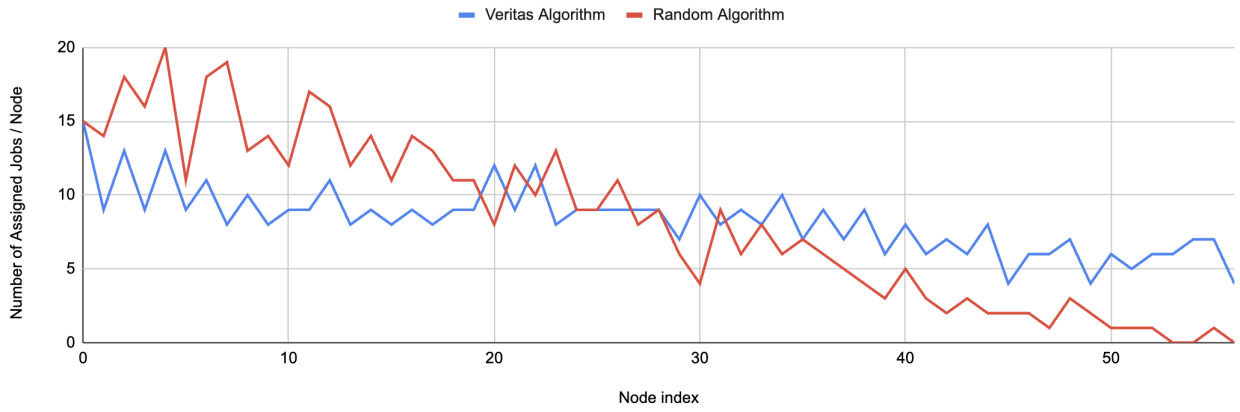

Figure 9. Node Selection Algorithm—Variant 3.

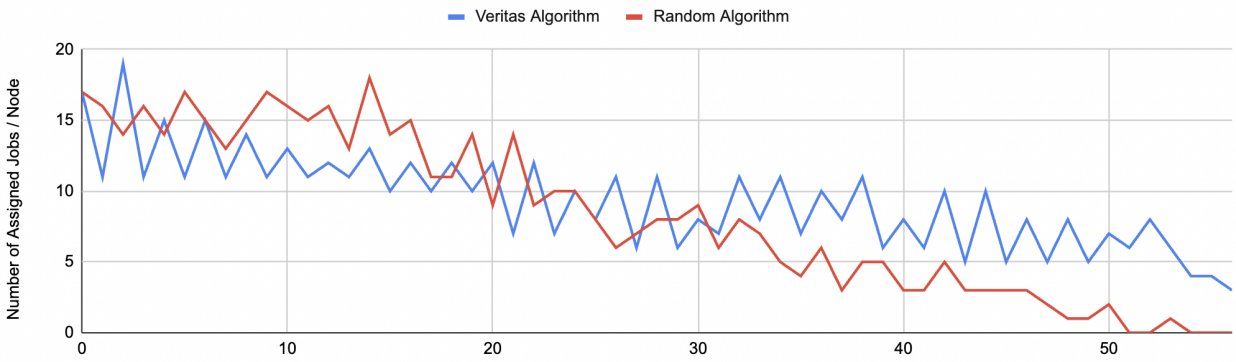

Figure 10. Node Selection Algorithm—Variant 4.

6. Discussion and Threats to Validity

6.1. Reflection on Experimental Results for Veritas and Chainlink Siimulations

As demonstrated in Section 5, our Veritas simulation answers **RQ1** by having a sidechain that aggregates results transparently and sends a single result back to the Ethereum main net. Now, according to the Chainlink white paper [42], it mentions 'commit-reveal schemes' when nodes submit requests; that means that we can safely assume that if the Chainlink simulation has this functionality, submitting a result would consume at least as much gas as the Veritas design. Also, aggregation, as per Chainlink [42], is performed on-chain, on the Ethereum chain, that is, meaning that all nodes involved in a data fetching job would have to send their data to the Ethereum main chain independently to be aggregated on a Chainlink smart contract there.

As per Equations (10)–(14), we answer **RQ2**, where it demonstrates the advantage Veritas has in terms of efficiency fulfilling real-time data fetching jobs. This is because the aggregation from the nodes on the side chain has a gas cost substantially lesser than that of the gas cost on Ethereum [45]. In addition, the number of data points sent back to the Ethereum main net is substantially lesser since the number of requests in the case of Veritas is multiplied by factor V which is a subset of the data points that deviate more than threshold Dev, thereby greatly reducing the cost of the job.

In addition, since the gas price on a blockchain network is directly correlated to the number of nodes on the network and network congestion, for any sidechain or Layer-2 blockchain, the gas price is lower than that on an Ethereum main net and, therefore, transaction fees is lower. Estimating gas fees is outside the scope of this paper. However, for reference, the transaction fee on Polygon (Layer-2) for making a simple transfer of the USDT cryptocurrency versus the transaction fee on Ethereum on 4 December 2023 is USD 0.006 vs. USD 2.73, respectively. This is a 455x difference in transaction fees [45].

This price fluctuates violently on both networks according to the number of operating nodes, ongoing transactions, and network congestion. At times, the transaction fees on the Ethereum network can be lesser than on the Polygon network, but this happens rarely and is often due to ICOs or, more recently, the deployment of a Dapp game on the Polygon network that consumes so many resources and fees skyrocket.

Security and Verifiability Advantage

Chainlink [42] mentioned that they plan, in the medium and long terms, to adopt a new design for a decentralized oracle system that aggregates off-chain. We do not know whether this is currently implemented and deployed or not, but it is safe to say that this proposal solves the previous problem since there will also be only one call to the Ethereum blockchain, but jeopardizes the advantage it had where the aggregation of the results was previously visible on the Ethereum blockchain and verifiable since the aggregation was performed through a Chainlink particular aggregation smart contract.

Veritas's advantage here is that it is a side chain, which means that every interaction between the oracle nodes and the VeSC that orchestrates the assignment of jobs and the

collection of responses is transparent and verifiable in the side chain public ledger; a clear advantage over off-chain aggregation, even if off-chain was a more straightforward design. To this end, the Oracle problem cannot be solved by processing data outside a blockchain since it provides the proven and tested consensus mechanism to verify each interaction with the public ledger.

6.2. Node Selection

In Section 5.2.3, we present the results of the two experiments we conducted in an attempt to show that Chainlink's method for selecting nodes based on bidding and then randomly selecting and considering only reputation presented two challenges:

1. Possible selection of malicious nodes due to the incentive to underbid in the bidding process for a node to be selected.
2. The absence of load balancing leads to long-standing nodes being assigned more jobs over time, jeopardizing the integrity of the selection process since those nodes statistically have a greater chance of being assigned.

We believe, through the experimental evaluation and result analysis performed, that we also resolved **RQ3** addressing the issue of the method proposed by Veritas increasing the security aspects of the Oracle system by making sure that nodes do not have any input in the selection to accomplish jobs, except by reputation, since underbidding could undermine the protocol for selecting nodes and thereby compromising the data input into the system.

6.3. Threats to Validity

We realize that there exist several threats to the validity of our proposed model, and in this section, we present them along with proposed solutions where possible.

1. Scalability issues due to node selection and possible congestion: Realizing that any Layer-2-based solution aims to provide scalability to a Layer-1 blockchain network, we also note that with an increasing number of nodes throughout the network, possible congestion could jeopardize the gas price of the side chain, gas consumed to select a node, and also the time required to serve a request due to the potential longer duration it could take to select a node.
2. Absence of timers and triggers: Any smart contract on a blockchain is triggered by an interaction from an externally owned account (EOA), as mentioned earlier. In our design, the VeSC aggregates responses from nodes, and when a minimum number of nodes have at least responded, the result is aggregated and sent back to the requesting smart contract. However, there are scenarios where a node might not respond and the minimum quorum may not be reached. In such cases, a trigger is needed to alert the smart contract to halt the request and not wait any longer, possibly refunding the requested contract. Additionally, there can be cases where a node responds, but network issues, such as dropped packets or unstable connections, can hinder the arrival of responses in a timely manner. The question arises: How long should the contract wait? In the current blockchain and smart contract development, there is no possibility of executing smart contract functions autonomously using an internal timer; there needs to be an external actor.
3. Lack of optimization to the smart contract VeSC. Currently, the gas units consumed to perform the required operation is high due to the computation and storage requirements of handling incoming jobs, maintaining node information and assigning tasks to nodes. This needs to be further optimized to minimize gas consumption and facilitate better operational costs. Otherwise, the cost gains could diminish as nodes scale up.

7. Conclusions and Future Work

In this paper, we contribute a new approach to solving the 'Oracle problem', which represents a significant challenge in the wide-scale adoption of blockchains in real-world

applications. We build on previous accomplishments in this space and attempt to resolve issues regarding very high transaction costs and obstacles that prevent blockchain applications from employing real-time data monitoring without compromising the security and integrity of the blockchain. We accomplish this by designing a distributed Oracle network that is itself a sidechain, connected to significant blockchains (in our case, Ethereum) via network bridges, and having the sidechain serve job requests securely and cost effectively.

We show that this approach is cost-effective and secure; it can validate our proposed model via a simulation using simulated blockchains. We attempt to compare our findings against those of one of the leading oracles in the space, Chainlink. We demonstrate that our proposed model tackles the cost problem much more efficiently and prove that our method for node selection provides better results in preventing malicious nodes from joining the network and thereby risking the network's security by introducing malicious data.

In the future, we plan to build upon our proposal by exploring the threats to validity we raised and attempting to resolve them. We mainly want to explore scalability issues with our design and analyze the effect of growing side chains on serving requested jobs. We also want to address the issues of external triggers and timers, which could be a bottleneck and potentially break the distributed nature of the Oracle system because they may introduce single points of failure. Other issues regarding synchronizing response times and accounting for network issues must be explored, experimented with, and analyzed.

In addition, we would like to apply the Veritas oracle system to serve a Web3 application that sources data from sensors and various IoT devices to demonstrate the validity and efficiency of the decentralized Oracle system and prove its capability to provide continuous and secure data streams and make IoT-Web3 applications a reality.

Author Contributions: Conceptualization, M.M.S.; methodology, M.M.S.; software, M.M.S.; resource, D.S.; investigation, M.M.S. and D.S.; validation, M.M.S., D.S. and A.A.S.; writing—original draft preparation, M.M.S.; writing—review and editing, M.M.S., D.S. and A.A.S.; supervision, D.S. and A.A.S.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hafid, A.; Hafid, A.S.; Samih, M. Scaling blockchains: A comprehensive survey. *IEEE Access* **2020**, *8*, 125244–125262. [CrossRef]
2. Zhang, W.; Anand, T. Layer 2 and Ethereum 2. In *Blockchain and Ethereum Smart Contract Solution Development: Dapp Programming with Solidity*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 341–378.
3. Sivaraman, V.; Venkatakrishnan, S.B.; Ruan, K.; Negi, P.; Yang, L.; Mittal, R.; Fanti, G.; Alizadeh, M. High throughput cryptocurrency routing in payment channel networks. In Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), Santa Clara, CA, USA, 25–27 February 2020; pp. 777–796.
4. Beniiche, A. A Study of Blockchain Oracles. *arXiv* **2020**, arXiv:2004.07140. http://arxiv.org/abs/2004.07140.
5. Caldarelli, G. Overview of blockchain oracle research. *Future Internet* **2022**, *14*, 175. [CrossRef]
6. Basile, D.; Goretti, V.; Di Ciccio, C.; Kirrane, S. Enhancing blockchain-based processes with decentralized oracles. In *International Conference on Business Process Management, Proceedings of the Business Process Management: Blockchain and Robotic Process Automation Forum, Rome, Italy, 6–10 September 2021*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 102–118.
7. Ezzat, S.K.; Saleh, Y.N.; Abdel-Hamid, A.A. Blockchain oracles: State-of-the-art and research directions. *IEEE Access* **2022**, *10*, 67551–67572. [CrossRef]
8. Al-Breiki, H.; Rehman, M.H.U.; Salah, K.; Svetinovic, D. Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access* **2020**, *8*, 85675–85685. [CrossRef]
9. Sriman, B.; Ganesh Kumar, S.; Shamili, P. Blockchain technology: Consensus protocol proof of work and proof of stake. In *Intelligent Computing and Applications: Proceedings of ICICA 2019*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 395–406.
10. Kapengut, E.; Mizrach, B. An event study of the ethereum transition to proof-of-stake. *Commodities* **2023**, *2*, 6. [CrossRef]
11. Nofer, M.; Gomber, P.; Hinz, O.; Schiereck, D. Blockchain. *Bus. Inf. Syst. Eng.* **2017**, *59*, 183–187. [CrossRef]
12. Monrat, A.A.; Schelén, O.; Andersson, K. A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access* **2019**, *7*, 117134–117151. [CrossRef]
13. Extance, A. Bitcoin and beyond. *Nature* **2015**, *526*, 21. [CrossRef]

14. Zaghloul, E.; Li, T.; Mutka, M.W.; Ren, J. Bitcoin and blockchain: Security and privacy. *IEEE Internet Things J.* **2020**, *7*, 10288–10313. [CrossRef]
15. Anceaume, E.; Ludinard, R.; Potop-Butucaru, M.; Tronel, F. Bitcoin a distributed shared register. In Proceedings of the Stabilization, Safety, and Security of Distributed Systems: 19th International Symposium, SSS 2017, Boston, MA, USA, 5–8 November 2017; Proceedings 19; pp. 456–468.
16. Burkhardt, D.; Werling, M.; Lasi, H. Distributed ledger. In Proceedings of the 2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Stuttgart, Germany, 17–20 June 2018; pp. 1–9.
17. Gervais, A.; Karame, G.O.; Wüst, K.; Glykantzis, V.; Ritzdorf, H.; Capkun, S. On the security and performance of proof of work blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 3–16.
18. Zheng, Z.; Xie, S.; Dai, H.N.; Chen, W.; Chen, X.; Weng, J.; Imran, M. An overview on smart contracts: Challenges, advances and platforms. *Future Gener. Comput. Syst.* **2020**, *105*, 475–491. [CrossRef]
19. Kemmoe, V.Y.; Stone, W.; Kim, J.; Kim, D.; Son, J. Recent advances in smart contracts: A technical overview and state of the art. *IEEE Access* **2020**, *8*, 117782–117801. [CrossRef]
20. Vujičić, D.; Jagodić, D.; Ranđić, S. Blockchain technology, bitcoin, and Ethereum: A brief overview. In Proceedings of the 2018 17th International Symposium Infoteh-Jahorina (Infoteh), East Sarajevo, Bosnia and Herzegovina, 21–23 March 2018; pp. 1–6.
21. Wang, Z.; Jin, H.; Dai, W.; Choo, K.K.R.; Zou, D. Ethereum smart contract security research: Survey and future research opportunities. *Front. Comput. Sci.* **2021**, *15*, 152802. [CrossRef]
22. Zhang, W.; Anand, T. Ethereum architecture and overview. In *Blockchain and Ethereum Smart Contract Solution Development: Dapp Programming with Solidity*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 209–244.
23. Yan, S. Analysis on blockchain consensus mechanism based on Proof of Work and Proof of Stake. In Proceedings of the 2022 International Conference on Data Analytics, Computing and Artificial Intelligence (ICDACAI), Zakopane, Poland, 15–16 August 2022; pp. 464–467.
24. Dannen, C. *Introducing Ethereum and Solidity*; Springer: Berlin/Heidelberg, Germany, 2017; Volume 1.
25. Tikhomirov, S. Ethereum: State of knowledge and research perspectives. In *Foundations and Practice of Security, Proceedings of the 10th International Symposium, FPS 2017, Nancy, France, 23–25 October 2017*; Revised Selected Papers 10; Springer: Berlin/Heidelberg, Germany, 2018; pp. 206–221.
26. Zheng, G.; Gao, L.; Huang, L.; Guan, J. *Ethereum Smart Contract Development in Solidity*; Springer: Berlin/Heidelberg, Germany, 2021.
27. Pierro, G.A.; Rocha, H. The influence factors on ethereum transaction fees. In Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Montreal, QC, Canada, 27 May 2019; pp. 24–31.
28. Ray, P.P. Web3: A comprehensive review on background, technologies, applications, zero-trust architectures, challenges and future directions. *Internet Things-Cyber-Phys. Syst.* **2023**, *3*, 213–248. [CrossRef]
29. Liu, W.; Cao, B.; Peng, M. Web3 Technologies: Challenges and Opportunities. *IEEE Netw.* **2023** . [CrossRef]
30. Cao, L. Decentralized ai: Edge intelligence and smart blockchain, metaverse, web3, and desci. *IEEE Intell. Syst.* **2022**, *37*, 6–19. [CrossRef]
31. Ding, W.; Hou, J.; Li, J.; Guo, C.; Qin, J.; Kozma, R.; Wang, F.Y. DeSci based on Web3 and DAO: A comprehensive overview and reference model. *IEEE Trans. Comput. Soc. Syst.* **2022**, *9*, 1563–1573. [CrossRef]
32. Sheridan, D.; Harris, J.; Wear, F.; Cowell, J., Jr.; Wong, E.; Yazdinejad, A. Web3 challenges and opportunities for the market. *arXiv* **2022**, arXiv:2209.02446.
33. Provable White Paper. Available online: https://api-new.whitepaper.io/documents/pdf?id=Sk89Yopev (accessed on 30 December 2023).
34. Jauernig, P.; Sadeghi, A.R.; Stapf, E. Trusted execution environments: Properties, applications, and challenges. *IEEE Secur. Priv.* **2020**, *18*, 56–60. [CrossRef]
35. Zhang, F.; Cecchetti, E.; Croman, K.; Juels, A.; Shi, E. Town crier: An authenticated data feed for smart contracts. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 270–282.
36. McKeen, F.; Alexandrovich, I.; Anati, I.; Caspi, D.; Johnson, S.; Leslie-Hurd, R.; Rozas, C. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, Seoul, Republic of Korea, 18 June 2016; pp. 1–9.
37. Peterson, J.; Krug, J. Augur: A decentralized, open-source platform for prediction markets. *arXiv* **2015**, arXiv:1501.01042.
38. Zhang, F.; Maram, D.; Malvai, H.; Goldfeder, S.; Juels, A. Deco: Liberating web data using decentralized oracles for tls. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 1919–1938.
39. Adler, J.; Berryhill, R.; Veneris, A.; Poulos, Z.; Veira, N.; Kastania, A. Astraea: A decentralized blockchain oracle. In Proceedings of the 2018 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1145–1152.

40. Benligiray, B.; Milic, S.; Vänttinen, H. Decentralized apis for web 3.0. In *API3 Foundation Whitepaper*; 2020. Available online: https://api.semanticscholar.org/CorpusID:265430507 (accessed on 14 February 2024).
41. Pasdar, A.; Lee, Y.C.; Dong, Z. Connect API with blockchain: A survey on blockchain oracle implementation. *ACM Comput. Surv.* **2023**, *55*, 1–39. [CrossRef]
42. Breidenbach, L.; Cachin, C.; Chan, B.; Coventry, A.; Ellis, S.; Juels, A.; Koushanfar, F.; Miller, A.; Magauran, B.; Moroz, D.; et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chain. Labs* **2021**, *1*, 1–136.
43. Ganache | Overview—Truffle Suite—Trufflesuite.com. Available online: https://trufflesuite.com/docs/ganache/ (accessed on 29 December 2023).
44. Shah, J.; Dubaria, D. Building modern clouds: Using docker, kubernetes & Google cloud platform. In Proceedings of the 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 7–9 January 2019; pp. 0184–0189.
45. Available online: https://etherscan.io/ (accessed on 30 December 2023).