Verifiable Computation over the Helix Tangle

Gowri Sankar Ramachandran
University of Southern California,
Los Angeles, USA.
gsramach@usc.edu

Oliver Fohrmann
Helix Cognitive Computing,
Berlin, Germany.
of@hlx.ai

Bhaskar Krishnamachari
University of Southern California,
Los Angeles, USA.
bkrishna@usc.edu

May 5, 2019

Abstract

The cryptocurrency platforms such as BitCoin and Ethereum rely on Proof-of-Work (PoW) consensus algorithm. Following the PoW, the nodes that solve the computationally intensive cryptographic puzzle is given the opportunity to create the next block in return for an incentive. Such puzzle-solving nodes are called "mining nodes," and its owners' are referred to as miners. Mining nodes are powerful computation platforms with the ability to perform computationally-intensive tasks. Although contemporary PoW computations demand high computation resources, the outcome of the PoW is not practical as the result of the computation is not used for any other applications except securing the blockchain. Using the mining nodes to execute practical algorithms for applications such as artificial intelligence, protein folding, and video processing would increase the effectiveness of the mining nodes while securing the blockchain. However, the distribution of computation tasks and sensitive data among trustless mining nodes leads to security vulnerabilities. Besides, the lack of technologies for verifying the result of the computation performed by the mining nodes makes verifiable computation impractical. In this work, we formulate the verifiable computation problem, discuss the existing work, and propose a novel platform for verifiable computation for the Helix Tangle.

Contents

1 Introduction

Verifiable computation paradigm focuses on crowdsourcing computation platforms. It enables the application or task owner ("submitter") to off-load one or more computation tasks to compute nodes ("worker") to speed up the computation and to exploit the cheap and powerful compute nodes. When the compute nodes or workers successfully performed the computation, the results have to be verified by "approvers" or "validators" nodes. To verify the result, the "approvers," "validators" or "submitters" are not required to redo the computation since that would minimize the effectiveness of the computation-off-loading approach. The verifiable computation paradigm focuses on ensuring the correctness of the computation without redoing the computation at the "validator" nodes.

To create a trustworthy and reliable verifiable computation platform, it is important to fulfill the following requirements:

- Support for "submitter" or "task owners" to submit the computation task. This process involves selecting the desired computation model and compute nodes.
- Ability to dispatch the compute tasks and associated metadata including inputs and other configuration details to the compute nodes.
- Scheduling of compute tasks on compute tasks while satisfying the demands of the users.
- Verification of the computation result, without redoing the entire computation, to ensure that the compute nodes have performed their work honestly.
- Reputation management framework for rating the compute nodes and the task submitters.
- Payment handler to transfer the computation fee from user's account to the compute node's account.

The rest of the document introduces verifiable computation and discusses the state-of-the-art in verifiable before proposing a novel solution for verifiable computation on Helix Tangle.

2 What is Verifiable Computation?

Verifiable computation enables the application or task owner ("submitter") to off-load or outsource computation tasks to compute nodes ("worker") contributed by the community members. When the compute nodes or workers successfully finish the computation, the results have to be verified by "approvers" or "validators" nodes.

To verify the result, the "approvers," "validators" or "submitters" are not required to redo the entire computation since that would minimize the effectiveness of the computation off-loading approach. The verifiable computation paradigm focuses on ensuring the correctness of the computation without redoing the entire computation at the "validator" nodes.

3 Verifiable Computation for the IoT, Smart Cities, and Connected and Autonomous Vehicles Applications

The cloud platforms have been widely used for performing data analytics, image processing, and other machine learning and artificial intelligence applications in the context of IoT, smart cities, and connected and autonomous vehicle applications [1, 2]. We will some of the example applications where verifiable computation could be beneficial.

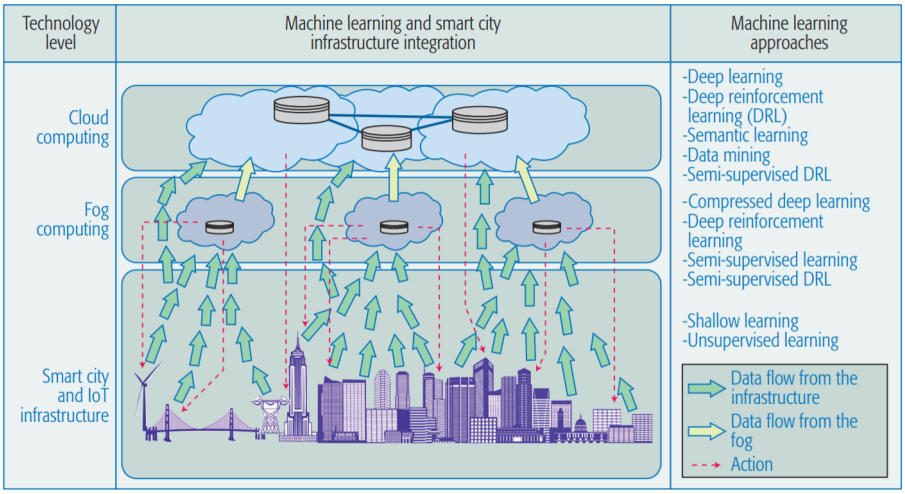

Figure 1: Application of data analytics for smart cities and IoT [3].

Data analytics for smart cities: An extensive collection of sensors and IoT devices are expected to generate a massive amount of data. To make sense of the data and automate the administrative and maintenance tasks, application developers are considering machine learning and artificial intelligence algorithms [3, 4]. Figure 1 shows the list of machine learning approaches that are beneficial for smart cities and IoT applications.

The verifiable computation platform can serve various applications by maintaining a standard set of tasks for data analytics applications. Users can select the desired machine learning or artificial intelligence algorithm and feed their inputs

to it. Unlike contemporary cloud platforms, the Helix verifiable computation focuses on scheduling the user's computation tasks on compute nodes provided by the community members.

3.1 Actors

Submitter: The node that submits the task to the network. This node is the entry point for the application developers.

Worker or Prover: The node that is part of the distributed network with powerful computation and storage capabilities, and is willing to outsource the resources for computation.

Validator or Approver: The node that validates the outcome of the computation. This node is not required to redo the computation, and it is assumed that these nodes are not as powerful as the worker nodes.

Fabric Initiator: The fabric initiator creates the fabric formation and may decide in this initial transaction (i.e., the fabric_root_index), what type of Fabric is hosted by specifying the fabric-type parameter.

- **Type A:** Mining-pool scheme (as described in 5.5 with some additions), the entity that submits the fabric initiation transaction has the complete authority to decide which compute tasks are accepted to the fabric, in which order they are processed and which entities may join the fabric and essentially has similar privilege to any mining pool leader.
- **Type B:** Validator based verification of results.
- **Type C:** Fully decentralized approach: Quorum-based verification of the results.

The freedom and the flexibility are proportional to the number of options that the system provides for the participants, which seems like a sound basis for an egalitarian system. Even though Type-A contradicts with some of the core values, it will be most likely be the most used system, especially by tech-institutes, as it provides best performance and control.

3.2 Requirements

The verifiable computation framework must fulfill the following requirements:

- Decentralization: The entire system must rely on a decentralized infrastructure to increase the trust and to avoid biases.
- Trust: The parties involved in the system must trust each other for the entire system to work sustainably.

- Security: The computation tasks and the results must be securely handled inside the system. The lack of security may prevent the end-users from joining the system.
- Cheat-proof guarantees: The owners of the compute nodes, and the task submitters may act maliciously. Section 4 presents the possible attacks.

The verifiable computation platform must be built with the above guarantees.

3.3 Related Work

This section reviews the literature on verifiable computing.

3.3.1 SETI@Home

Anderson et al. [5] contributes SETI @ Home to outsource the analysis of radio signals for a project that searches for extraterrestrial intelligence. Radio signals picked up by telescopes consist of signals from various source including TV and Satellites. To effectively dissect the signal, a fine-grained frequency analysis is required, and such an analysis requires enormous computation resource. SETI @ Home project allow any machine on the planet to join the SETI network and contribute computation resource in return for credit points. Users receive credits when they complete a task but how the credits are used is not clear.

SETI@Home follows a client-server model, wherein the computation tasks are distributed to the clients from a server. Malicious actors produced inconsistent or incorrect results to gain high ratings in SETI @ Home project [6]. Result verification through replication is used as a solution to guard the system from malicious workers. Following this approach, the same computation tasks are assigned to multiple clients, and the results from clients are compared to ensure correctness. When a quorum is reached on the results, the worker nodes are rewarded through credit points.

Zhao et al. [6] points out the weaknesses in the replication-based verification schemes. Worker nodes in the network collude and may return a same incorrect result to gain credit points without performing any computation. *Zhao et al.* [6] contributes Quiz mechanism to resolve the issues with the replication-based scheme. The Quiz mechanism inserts quizzes as part of the computation task, and when a client returns the result, the results of the quizzes are verified to ensure correctness since the wrong results to a quiz indicate suspicious behavior. To further strengthen the system, a trust-based scheduling scheme is proposed, wherein the tasks are allocated to trusted clients. In this approach, honest clients are selected based on their past performance and are given high preference during the scheduling phase.

3.3.2 Homomorphic Encryption

Encryption schemes are typically used to preserve sensitive information. Systems that use encryption schemes have to decrypt the data before performing any operations on the encrypted data. Homomorphic encryption schemes preserve privacy by performing computation on decrypted data [7]. In other words, the encrypted result of the computations on encrypted data will match the result of the computations performed on plain-text.

Homomorphic encryption schemes have been developed for a number of applications in the last two decades, but the practical use of such systems is still under scrutiny due to their resource demands [8].

3.3.3 Proof-based verification

The secure multi-party computation was introduced in 1982 by Andrew Yao [9] to allow two parties to perform computation without relying on third parties while keeping the local data or input secret [10]. Approaches such as zero-knowledge proof allow the worker and validator to verify the result without exchanging any information except the fact that the verifier knows the correct result [11]. Zero-knowledge proof, one of the interactive proof systems, has the following properties:

Completeness: The worker knows the truth, and he/she will convince the verifier eventually. Soundness: The worker can convince the verifier only if he/she tells the truth. Zero-knowledgeness: No information about the truth is exchanged between the worker and the verifier.

ZKSnark [12] is one of the practical implementations of Zero-knowledge proof, and it uses the libsnark library. To use the zero-knowledge proof mechanism, the computation task has to be translated into the right format using the following steps:

- **Computation → Circuit:** The computation task has to be converted into a sequence of logic expressions, which are represented as logic gates. This process is also known as "flattening" process.
- **Circuit → Rank-1 Constraint System (R1CS):** The flattened expressions should be converted into R1CS format. The circuits are converted into a tuple (a,b,c) which results in a solution s. For the R1CS to be satisfiable [13], the s must satisfy the equation: s . a * s . b - s . c = 0.
- **R1CS → Quadratic Arithmetic Program (QAP):** R1CS expressions are converted into polynomials, which are then used for verifying the proof of the computation at random points [14]. In other words, the verification of individual outputs from each and every logic gate is cumbersome. Gennaro et al. [14] contributed QAP to minimize the verification complexity, wherein the prover can check the constraints at any random point in the polynomial.

The above steps are essential for each computation problem in the ZKSnark system. Besides, the computation currently supports basic arithmetic operations (+, - , *, /), exponent function with constant power, and assignment operator. Loop operations and comparison operators are not currently supported. Although the ZKSnark is promising, it lacks practical tools and frameworks for generic computation problem. LibSTARK is another implementation of the proof-based verification scheme, which is used by ZK-STARK [12].

3.3.4 TrueBit

TrueBit [15] presents trustless smart contracts to enable secure computation on the blockchain. In PoW-based blockchain systems, each node in the network executes the smart contract to verify the result, and the nodes are not rewarded for their verification. The node that solves the cryptographic puzzle gets an incentive since it is authorized to create the next block, whereas all the other nodes verify the integrity of the block to make sure they are attaching themselves to an invalid or malicious chain. TrueBit proposes a novel approach, by which a set of special nodes are elected to perform the verification instead of asking all the validator nodes to expend resources for the computation [15]. TrueBit injects faults to the computation to check the behavior of the validating nodes. Nodes that fail to report the "forced errors" are penalized for their dishonest behavior. This feature enables TrueBit to remove dishonest nodes from the network.

3.3.5 Perlin

Perlin [16] is a decentralized compute platform on a distributed acyclic graph (DAG) ledger technology. In particular, Perlin uses the Avalanche protocol [17], which is a novel DAG-based ledger based on a concept called "metastability." Metastability allows the nodes in the network to come to a consensus in a series of rounds, in which the nodes that are part of the quorum are repeatedly voting to decide on the stable state for a given event. Perlin uses Avalanche protocol to build a novel decentralized compute layer. The key contributions of Perlin include Sybil-resistant identity management through a PoW scheme, whose complexity is selected by the participants in the network through quorum-based voting. Although Perlin announces itself as a decentralized compute layer, it lacks information about the system. Moreover, Perlin's model assumes that the user executes the computation again to ensure correctness, which makes the verifiable computation weaker, as the user can execute the computation on his/her machine in the first place without off-loading to another machine.

3.3.6 Hardware-based Approaches

Ekiden [18] is a privacy-preserving computation platform for the execution of smart contracts. Intel SGX is used for executing the computation in a cryptographically secure execution environment. Similarly, *Ohrimenko et al.* [19]contributes a framework for running machine learning applications in a trusted hardware environment. Such hardware-based approaches rely on the security mechanisms provided by the underlying hardware and perform all the computation inside the secure environment to preserve the privacy. Only the hardware with the support for SGX can participate in the network, which is a major limitation of hardware-based approaches.

This section discussed the state-of-the-art in verifiable computation. We will discuss the attacks in the next section.

4 Attacker Models

In verifiable computation platform, nodes and the users can indulge in the following activities to impact the normal operation of the system.

- **A1:** Malicious nodes that return an incorrect result either regularly or sporadically.
- **A2:** Colluding cheaters jointly submit the wrong result to fool the consensus layer. Following the BFT consensus model, the system requires 2/3rd of the majority to attack the system.
- **A3:** Smart malicious node may leave the system for a period and rejoin again with a different identity to start gaining reputation.
- **A4:** Buggy computational task wasting the resources of the compute nodes.
- **A5:** Nodes copy the right result from one of the nodes in the network to gain incentive without doing any computation.
- **A6:** The owners of the compute nodes can submit tasks frequently to gain reputation.
- **A7: The reputation metric** If reputation is defined as a metric that maps proportionally to the complexity of a compute task, an adversary could exploit this by maximizing reputation gain through auto-submission of specifically optimized compute tasks to a specifically optimized engine (such as an ASIC).

5 Overview of HelixMesh Protocol

The verifiable computation framework runs on top of the HelixMesh, which is referred to as Helix Tangle in this article. HelixMesh is a double consensus framework inspired by MeshCash framework [20]. The consensus process consists of off-chain and on-chain layers with complete flexibility to choose any protocols at both layers. Besides, this architecture provides the ability to run either permissioned or public protocols. In particular, the protocol provides a right balance between speed and throughput through the combination of on-chain and off-chain consensus protocols.

The on-chain protocol, referred to as Tortoise protocol, runs a DAG-based consensus protocol, while the off-chain protocol, referred to as Hare consensus protocol. On the one hand, the hare consensus protocol runs slowly and may be slow in confirming the transactions, but it acts as an oracle and can predict the outcome and performance of Tortoise. On the other hand, the on-chain protocol is self-contained and make all the consensus decisions using the local state information. We refer the reader to the following link - https://hlx.ai/files/HelixMesh_V1_2019_04_09.pdf) - to learn about the HelixMesh proposal and the necessary details of Tortoise and Hare consensus protocols.

6 Approach

Figure 2 shows the building blocks of verifiable computation. We propose the following approach to solve the open problems in verifiable computing paradigm.

- **Joining Process:** The compute nodes are classified into different clusters, referred to as fabrics. Each fabric is mapped to a resource tier based on the resource capacity of the compute nodes. We employ a resource testing process to identify the resource capacity of the compute nodes.
  - The fabric initiator creates the fabric formation and may decide in this initial transaction (i.e., the fabric_root_index), what type of Fabric is hosted. Different models of fabrics are discussed in Section 7. Each fabric has a leader, who is responsible for managing the compute nodes. The leader selection protocol is responsible for electing the leader, which depends on the fabric model (see Section 7).
  - The fabrics layer consists of a leader-election protocol, which selects the leader based on the fabric model. The leader is responsible for accepting the compute tasks, submitting the result of the compute task, collecting the payment, and managing the compute nodes.
  - The software for compute nodes includes the leader election protocol, payment handling mechanism, resource profiler, and a local scheduler would be provided by the Helix foundation.

Figure 2: Building Blocks of Verifiable Computing.

  - The resource test results are submitted to the fabric leader, where the results are compared against a resource model. The fabric leader places the compute node to one of the fabrics.
  - The compute nodes are placed on fabrics based on the outcome of the resource test.
  - The compute nodes are required to consistently contribute the same amount of resources to the system throughout its lifetime to retain its position in the resource tier. Nodes that wants to modify the resource capacity has to leave the system and rejoin again as a new compute node with different resource capacity. The resource testing process is executed sporadically to ensure that the nodes are actively contributing to the promised resources.

- **Compute Task Submission Process:** Users should have an account in the Helix network with sufficient Helix tokens to submit a compute task. The computation package consists of the code, input, and metadata such as the execution script, wallet address, and the resource requirements.
  - When the user submits the compute task, the system receives hundreds

of MegaBytes of data. As the submission of compute task increases, the bandwidth requirement for propagating the task package would increase. Thus, it is important to use storage nodes for saving compute packages following the IPFS storage model.

- Although the security and privacy are guaranteed through the built-in security mechanisms of the Tangle; the user may want to delete the compute task after receiving the desired result from the system. Support for removal of compute tasks and inputs may be desired.

- **Scheduling of compute tasks:** When the user submits a compute request to the Helix network, the task has to be scheduled for computation in one or more fabrics. We schedule the compute task on two fabrics using one of the following two approaches:

  - **Model A:** When the user submits the compute task, the metadata associated with the task is sent to all the fabrics in the resource tier selected by the user. Fabric leaders for the current epoch analyze the metadata and submit a bid for the computation task. User goes through the list of bids and selects two fabrics from the list for the computation.

  - **Model B:** When the user submits the compute task, he/she agrees to pay $payment^{max}$ for the fabrics that perform the computation. The consensus layer randomly selects two fabrics for the computation. The user has to pay the fee charged by the compute nodes, and it is important to ensure that the $payment^{max}$ is selected reasonably for a given compute task. The system can recommend payments based on the metadata.

- **Verification Process:** The fabrics perform the computation using the compute nodes that are part of their fabrics. It is assumed that all the compute nodes in the fabric performs the computation, and each fabric should have a minimum of N^{min} nodes to qualify for performing the compute tasks. Verification process starts within the fabric and ends at the consensus layer as follows:

  - The fabric leader dispatches the task to all the compute nodes in its cluster. Nodes perform the computation and return the result along with the resource usage to the fabric leader for the verification. The fabric leader collects the results and submits them to the consensus layer for the final confirmation. Each fabric involved in the computation is expected to submit the results to the consensus layer.

  - The consensus layer collects the results from the fabric and comes to an agreement on the results, and also compares the resource usage of all the computing nodes to check if there are any discrepancies.

- **Payment and Reputation:** When the computation is verified successfully, the payment and the reputation handler will receive a notification to do the following:
  - **Payment Handler:** Based on the payment mode selected by the user at the time of submission, the system handles the payment. For the bidding model, the system charges the amount bid by the user, and the user is charged based on the resource usage if the system scheduled the computation on the fabrics. The user's wallet address and the fabrics' wallets address maintained by the payment engine, and it transfers the fee from the user's account to the cluster's account.
  - **Reputation System:** The compute nodes, and the user must be rated after the computation is successfully performed on the system. The compute nodes are rated positively when the user does not challenge the results.

7 Joining Process

The community members including industries, technology startups, and research laboratories are expected to contribute resources to the Helix Tangle for verifiable computation. This section discusses how the compute nodes can join the network and provide computing resources.

The compute nodes are expected to undergo a resource test when joining the system. The resource test is performed using a set of test tasks maintained by the system. Resource hardness properties can be used to accurately estimate the resource capacity of the compute nodes [21]. The three key metrics of the compute tasks are computation time (in seconds), memory usage (in bytes) and the size of the task (in bytes) [22]. Creating a set of compute tasks combining these metrics would cover a broad spectrum of compute tasks.

The compute node executes the task and report their resource usage to the consensus/verification layer. We assume that the fabric leader is responsible for the verification results to minimize the resource overhead of the Helix Tangle. The resource test results are analyzed, and the compute nodes are placed in a suitable fabric. The critical problem here is how the Helix Tangle can trust the resources reported by the compute nodes.

The fabrics are classified into different categories based on the resource capacity as high, medium, and low. These categories are not fine-grained (at the moment), as the fine-grained classification may make it difficult to map the compute nodes to fabrics accurately.

7.1 Identity Management

Helix's verifiable computation enable the community members to join the platform and contribute their resources for the computation. To make sure that the nodes are behaving honestly, it is important to create an identity management. We consider a hybrid identity management mechanism in which, the Helix bootstraps the verifiable computation platform by using the nodes maintained and managed by the *Helix* foundation. Since the Helix foundation acts in the best interest of its users, the users can reliably off-load their computation to Helix platform. But, our approach transition from a permission setup (managed by Helix foundation) to a public setup through a combination of chain of trust and collateral-based trust mechanism. In a nutshell, our approach works as follows:

- Helix launches the verifiable computation platform by using the computation nodes maintained and managed by the Helix foundation.
- The public nodes can then join the Helix eco-system by placing a slashable collateral. From this point on, both the nodes maintained by the Helix nodes and the public nodes are available to execute computations submitted by the users.
- Whenever an user submits a computation task, it is scheduled on two computation nodes. One of the computation nodes is selected from the cluster managed by the Helix computation nodes and another node is randomly selected from the public nodes. This form of scheduling allow the public nodes to gain reputation since the results produced by the public node can be compared with the permissioned node maintained by the Helix foundation.
- As the public nodes gain rating, they are elevated to the level of permissioned nodes maintained by the Helix foundation. Once a sufficient number of public nodes become a part of the Helix verifiable computation ecosystem, the permissioned nodes managed by the Helix foundation can be completely removed.

7.2 Fabric Formation and Management

The system starts with no fabrics at its disposal. Any entity capable of interfacing with the Helix Protocol may initiate a fabric formation by submitting a specialized transaction to the protocol-layer, this transaction includes metadata about various configuration parameters that are crucial for maintenance of fabric. The metadata should include fabric-type, consensus-type, resource-type, specific work type, resource-tier, scheduling-model, bidding-model, and the optional parameter: access-policies. If no access-policies are specified, the fabric participates in the automated joining process, i.e., performance-based sampling from the weights of the applicant's queue.

As the new compute nodes join the system, the fabrics are created before assigning the compute nodes to them. Each fabric requires a minimum of N^{min} compute nodes to qualify for the computation. Furthermore, the maximum number of compute nodes in the fabric is defined as N^{max}. N^{max} should be chosen such that byzantine failures between compute-nodes are tolerated, that is $N^{max} = 3f+1$. Typically, this means N^{max} is an even number, but in the case of voting schemes, as seen in e-governance, it may be reasonable to tolerate odd numbers too. In that case, N^{max} should be chosen, such that $N^{max} = 3f+1$ & $N^{max} \bmod 2 = 1$.

Within each fabric, we execute a fabric management framework, which consists of a leader-based BFT consensus algorithm along with a peer handler to orchestrate the interactions between compute nodes. The leader election process can be handled through a round-robin scheme or a token-passing scheme. At each epoch $\mathcal{E}_i$, a new leader is elected within the fabric to manage the compute nodes. The fabric leader is in charge of bidding for the compute tasks, monitoring the compute nodes, delegating the compute tasks, and distribution of payments to the individual compute nodes.

7.3 Sporadic Resource Testing for Liveness

The compute nodes may not actively participate in the computation process all the time, which may lead to a lack of compute devices. To encourage participation and to reward the consistent compute nodes, the system regularly performs resource tests and rewards the active nodes. The compute nodes that are active in the resource testing phase undergoes an evaluation of the resource capacity to ensure that the nodes are consistently contributing the resources (according to their resource tier). The verification layer positively rates the nodes that add constant resources for their continued and honest support. However, the nodes that do not participate in the sporadic resource testing or the nodes that participate but their resource contribution is different from the initially agreed resource contribution are not positively rated.

The sporadic testing adds randomness to the testing process as the periodic testing may provide an opportunity for the compute nodes to learn about the testing phase. The resource testing must be done unpredictably since the compute nodes come online only for the resource testing to gain reputation.

The system must maintain a collection of compute nodes with varying resource demands. Whenever a new compute joins the system, a set of compute tasks are executed on the new machine to test their resource capacity before assigning compute nodes to fabrics.

7.4 Need for an Escrow Deposit

The truthful behavior of compute nodes are essential for the verifiable computation platform as this would encourage the users to submit computation tasks to the system. However, the compute nodes explore methods to increase their pay-off. Some of the potential cheating methods presented in Section 4. An escrow deposit would prevent the nodes from cheating and other malicious activities as the nodes would lose their deposit when the Helix Tangle identifies them. When a compute node is found to be malicious, the money held in escrow is slashed by the Helix Tangle, and it will subsequently be used for rescheduling the computation (without charging the user).

7.5 Anonymous Resource Testing as a Challenge

From the compute node's point of view, the resource testing could be made anonymous. In other words, the compute task associated with the resource testing should not be distinguishable from the compute task submitted by the user. Such a mechanism would allow the Fabric Leader not only to catch the cheating nodes that copy the results from other nodes but also ensures liveness. TrueBit has the concept of forced error which forces the solver (compute nodes) to produce wrong results to verify whether the challengers are actively checking the outcome of the computations. As an incentive, TrueBit pays a jackpot to the challenger that caught the forced error. Similarly, the resource testing can happen at any time, and the compute nodes that undergo resource testing and produce the desired result using the right amount of resources are positively rated while the inconsistent compute nodes are downrated with additional penalties for misbehavior.

7.6 Requirements Validation

This section validates how the requirements listed in Section 3.2 are handled in the joining process.

- Decentralization: Can the joining process be made fully decentralized? Will this be handled by a special set of nodes? An alternative approach would be to follow bitcoin's mining pool [23] alike approach wherein the community members form the pools.

- Trust: Fabrics are trusted based on the behavior of the compute nodes. Each fabric may consist of a collection of honest and dishonest nodes. The dishonest nodes may negatively impact the rating of fabric, which in turn would affect the honest nodes from gaining incentives. Compute nodes gain additional incentive when they are elected as a fabric leader, where the selection protocol considers the rating/reputation of the computation as one of the parameters.

- Security: The resource testing uses a set of compute tasks with known resource estimates. All the communication between the core layer and the compute nodes must be secured. We assume that the security primitives provided by Tangle are sufficient to secure the communication.

- Cheat-proof operation: Compute nodes join the system based on the outcome of the resource testing process. How do we ensure that the reported resource usage is the true representation of the resource used by the node?

7.7 Alternative Approach to Fabric Formation

The joining approach based on resource testing introduces significant communication and management overhead. The Proof-of-Work (PoW) blockchain platforms rely heavily on the mining pools for the block creation. The nodes that want to make a steady income join a mining pool, wherein the rewards are shared among the members of the pool for their contribution [23]. Following a similar approach, the compute nodes can form the computation pools (or fabrics). Besides, the compute nodes can be self-managed by the fabrics. Such an approach allows the compute nodes to easily move between different fabrics when the fabrics contain a high-number of dishonest nodes, or the rewards are insufficient in a given fabric. This approach delegates the management activities to the pools as in the case of BitCoin and Ethereum.

8 Computation Task Template

The core aim of the system is to enable users to off-load computation tasks to the Helix verifiable computation platform. For the system to accept and schedule the compute task, a template should be provided to the user. Besides, the template should be generic enough to minimize the preparation overhead for the user. In other words, the supported template should not restrict the user from using any programming language.

We propose a container-based computation package in this work to minimize the preparation overhead and to accept the computation of all types. For the computation to be performed on the Helix's verifiable computation platform, the user must submit the following information to the system as a self-contained package:

- Compute task: The program that has to be executed.

- Input file: The input for the compute task.

- Meta data: Other information essential to schedule and execute the compute task on the system.

The tasks assumed to be uploaded to the storage nodes, and the pointer to the storage goes to the scheduling for further processing since storing the compute tasks on the user's platform not only increases the storage overhead but also requires an incentive mechanism for rewarding the storage nodes.

8.1 Submission of Spam

A malicious user may submit spam (malicious) task to mount a DoS attack on the system. The code cannot be executed at the time of reception to ensure that it is not spam since it depletes the purpose of verifiable computation. An escrow deposit along with a threshold that defines either the maximum number of steps executed or the maximum amount of resources would prevent the malicious user from wasting the computing resources. The compute platform should stop the execution when the given compute task is not completed within a given threshold.

8.2 Classification of Compute Tasks

We don't specify the types of applications that can be executed on the Helix's verifiable computation platform. The system may set an upper limit on the computation demand. As long as the submitted application task executes within the prescribed upper bound, the Helix Tangle successfully executes the compute tasks on the verifiable computation platform. A policy engine driven by a machine learning algorithm can be considered to intelligently schedule computation based on the resource demands, sensitivity, and the application type.

9 Scheduling of Compute Tasks

Figure 4 shows the interaction flow when the user submits a task for the computation.

When the user submits the task to the compute nodes, the scheduling is handled by a distributed scheduling subsystem, which runs alongside the consensus engine. The distributed scheduling subsystem receives the request and checks the scheduling policy field, which consists of the type of the scheduling policy selected by the user. The user can choose between "bidding" or "delegate scheduling to Helix." The process for scheduling changes based on the choice selected by the user.

Bidding Approach: The user gets to choose the fabric based on the bids submitted by the compute nodes. When the user decides to select the best bid for the computation task, the scheduling layer forwards all the metadata to the fabrics in the resource tier selected by the user. Note that the user is expected to select a resource tier when he/she submits a compute task to the system. The tier can be "high", "medium", or "low". For example, when the user selects a "medium" tier,

the metadata along with the reserve price for "medium" tier is sent to all the fabrics in the "medium" tier. Fabrics, i.e., the fabric leader of the current epoch processes the metadata and submits a bid for the compute task. The user receives the bids from the fabrics and selects fabrics for the computation, where $\mathcal{F}$ is the number of fabrics selected for the computation (see 9.2). *The critical question is how does the compute nodes know the fair-market-value for a given computation task.*

Scheduling By Helix: When the user delegates the scheduling to the Helix node, the distributed scheduling subsystem processes the metadata and randomly selects $\mathcal{F}$ fabrics for the computation. Note that the user is expected to submit the number of fabrics when submitting the compute tasks. A registry is maintained at the Helix system that consists of a list of fabrics along with their resource tier. A distributed random generator such as RANDHOUND and RANDHERD are considered for the selection of fabrics. However, the actual implementation may consider other approaches for the selection of fabrics based on the performance and unbiasedness. It is crucial to guarantee fairness when selecting computation nodes using a distributed random generator. A formal proof is necessary to assure the randomness when Helix schedules the computation.

9.1 Payment Agreement at Submission Time

On the one hand, the user does not have sufficient knowledge to quote a reasonable price for the task at the time of submission. On the other hand, the compute nodes do not want to perform the computation when the pay is unreasonable for the computation. For the system to operate sustainably, both parties should be satisfied with the service provided by the Helix's verifiable computation platform. Thus, it is important to clearly define how the costs are calculated for both the user and the compute nodes.

Expecting the user to quote a reasonable price for the computation is a risky process. Instead, the system can calculate the payment by splitting the computation into either time units, the resource consumption or even the combination of those two. Such a model would let the user select the pay per time unit or resource usage instead of quoting the actual price. Based on the resource usage and the time taken for execution, the system can estimate the actual cost and deduct the money from the user's wallet.

9.1.1 Model A

When the nodes bid for the computation, it would only suggest the price per time unit and/or resource usage. The user chooses the compute nodes based on the received bids.

9.1.2 Model B

When the user submits the task, the system reports the price per time unit and/or resource usage to the user along with a contract that explains the payment terms and condition.

9.1.3 Is There a Need for Escrow Deposit?

An escrow mechanism can be added at the early stages to make sure that the compute nodes paid from the user's wallet. The users' would gain reputation based on their behavior. For the users' with high reputation, the escrow mechanism can be made optional.

9.2 Number of Fabrics For The Computation

In practice, the user can select just one fabric for the computation. Scheduling the compute task on more than one fabric provides assurance, and the results of the computation can be compared to ensure correctness. From the user's perspective, the computation cost increases with the increase in the number of fabrics involved in the computation. When the system selects the fabrics without user's consent, the user may have to pay whatever the system charges. To increase the user's satisfaction and to provide transparency, the selection of the number of fabrics should be done by the user. As discussed in Section 9.2, our platform initially schedules the computation on two nodes (one each from permissioned Helix cluster and public) to allow the public nodes to gain reputation.

9.3 Validation of the Requirements

The scheduling subsystem must guarantee the requirements listed in Section 3.2. We will validate the requirements below:

- Decentralization: Although the scheduling subsystem is decentralized in Model B (Scheduling By Helix), the distributed random generator may schedule fabrics unfairly and deprive some nodes/fabrics of performing the computation. In the case of Model A, the Helix Tangle acts as a middleware between the user and the fabrics.

- Trust: From the compute node's perspective, Model B must prove that the random selection of fabrics is made in a fair manner, whereas in Model A, the fair-market-value quoted by the user should not be unreasonable.

  From the user's perspective, the Helix's verifiable computation platform should aid the user to select a reasonable fair-market-value for the computation along

with a contract that clearly defines the prize fluctuations for both Model A and Model B.

- Security: The communication between the parties are secured using the Helix's underlying security primitives.
- Cheat proof operation: As long as the contract clearly defines the payment terms for the user, the payment for the transaction can be deducted from the user's wallet. For Model A, the compute node that bid for a task must accept the computation when the user chooses it based on the bid. For both models, the computation task may exceed the quoted value, thus, the user should pay the actual cost, or the system should be designed to stop the computation when the payment limit is reached.

10 Verification of the results

The consensus layer is responsible for dispatching the computation tasks to the fabrics after the nodes are selected using the approaches presented in Section 9. Our system uses the following approach:

- The compute task is sent to the fabrics for the computation.
- Fabrics receives the compute package, which consists of a compute task, input, and the metadata describing the instructions for execution of the task.
- Fabric dispatches the package to a subset of compute nodes for execution.
- The compute nodes execute the task, and the execution framework consists of checkpoints for tracking the execution. At each checkpoint, the computation engine takes a snapshot of the stack and store it. After the end of execution, the snapshots are encrypted and stored in a distributed data storage.
- The results of the computation are submitted to the fabric leader, who then send the results to the users via the consensus layer.
- User receives the result, and check it. If the user is not convinced, he has a $\mathcal{C}_t$ period to challenge the results.
- When the user challenges the result, the consensus layer starts the verification process, by collecting the execution profile of the compute nodes from the fabric. Consensus layer queries the fabrics to provide access to the stack trace of the computation. After retrieving the stack trace, the consensus layer verifies the stack trace of all the compute nodes and ensure that there are no discrepancies. The consensus layer has to come to a quorum consensus on the results.

10.1 Need for a Second Price Auction

The price for the computation is calculated based on the execution time and resource usage. The compute nodes submit the resources used for the computation along with the results as this information is essential to compute the actual computation cost, to be paid by the user. This approach requires an algorithm for the calculation of the payment and the evaluation of resources. We propose a bidding mechanism to estimate the fee for the computation. Before going into the details of our solution, we will provide an overview of different bidding models relevant to our work.

10.1.1 Overview of Bidding Models

Regev et al. [24] proposed a computation marketplace platform called POPCORN[1]. POPCORN allows sellers to contribute computation resource through a web-application. The buyers of the computation platform, which is a software agent, select a seller using an auction scheme. POPCORN uses CPU time as the key metric to determine the price for the computation. Three different mechanisms are presented in the literature:

- **Repeated Vickrey Auction:** This approach uses a conventional second-price sealed Vickrey auction. The price/CPU-time defined by the sellers are used as the inputs. The computation task is given to the seller at the least price, but his incentives are based on the second highest price.
- **Simple Sealed-Bid Double Auction:** This approach allows the buyers to bid two values: high-price and low-price long with a rate of change. The price of the seller starts with the high price, and it is decreased following the rate of change until a buyer is found. Similarly, the buyer starts with a lower price and the price is increased until a seller with a comparable price is found.
- **Repeated Clearinghouse Double Action:** Unlike the previous approach, this approach matches more than one buyer-seller pair to find a match. Supply-demand curves are continuously updated at periodic intervals to identify the right candidate.

Recently, *Bhattacharya et al.* [25] present an extension to second price auction for auctioning energy to plug-in electric vehicles. This work presents the challenges in applying Vickrey auction for agents with unknown valuation functions. Two extensions to Vickrey second-price auction is presented in [25]:

- **Multi-level Second Price Auction:** Agents submit a set of prices for certain levels of energy. The aggregator calculates the valuation function (using piecewise linear approximation) and computes the optimal schedule for the agents.

[1] Verifiable computation platform developed in 1998 with interesting ideas.

- **Progressive Second Price:** Agents declare a two-dimensional bid (qk, pk), where qk is the amount of energy that the agent k is willing to buy at a per unit price of pk.

Luong et al. [26] present a deep learning approach for optimal auctions of edge resources for mobile blockchain applications. *Dütting et al.* [27] presents a deep learning architecture for optimal auctions, which is used by *Luong et al.* [26] to construct the neural network architecture. Neural networks allocate resources using the bidding valuations of miners as an input.

10.1.2 Vickrey Auction for Verifiable Computation

The node that used the least resources will be rewarded, but its reward is based on the second lowest resource usage. This type of auction is called second-price sealed bid auction or Vickrey auction. Following this scheme, the incentive is calculated as follows. The resource usage of Node A, $\mathcal{R}_{\mathcal{A}}$, is defined as:

$$\mathcal{R}_{\mathcal{A}} = \alpha$$

And, the resource usage of Node B $\mathcal{R}_{\mathcal{A}}$, is defined as:

$$\mathcal{R}_{\mathcal{B}} = \beta$$

Assuming that the node A used the least computation resource, the reward for NodeA is:

$$Reward(Node_A) = f(\mathcal{P}_u, \delta) \text{ where } \delta = \beta - \alpha$$

In the above equation, $\mathcal{P}_u$ represents the price per unit where the unit denotes either the execution time or the resource usage. The actual calculation of the payment will be described during the implementation.

10.2 Collusion Among Fabric or Compute Nodes

The user or the verification layer checks the results submitted by the compute nodes. Multiple compute nodes may collude and present wrong results, which may be wholly wrong and appears plausible. All the compute nodes participate in the verifiable computation process to increase its pay-off [28], in some cases, at any cost. It is therefore essential to devise a mechanism to discourage collusion in the verifiable computation platform.

Problem: In our approach, a compute node may receive the stack trace, results, execution time, and the resource usage from a colluding partner and submits to the Helix Tangle or the fabric leader.

Solution: Whenever the user challenges the result submitted by fabric or compute nodes, the challenge process verifies the submitted results along with the stack trace and the resource usage. The compute nodes with identical results and stack trace are maintained in a *suspect* queue as a pair (inspired from [29]). Whenever the user challenges the result, the participating compute nodes are checked against the *suspect* queue, and the node or pair of nodes that continuously appear in the *suspect* list is removed from the system with a penalty.

10.2.1 An approach to capture malicious actors

After the launch of the verifiable computation platform, the Helix Tangle should employ a trusted compute nodes/fabrics in combination with crowdsourced platforms and schedule computation tasks on both the trusted and the crowdsourced computation platforms. Such a mechanism would allow the compute nodes to gain reputation while providing a tool to identify dishonest compute nodes quickly. Since the trusted compute nodes owned by either the Helix Tangle or the Helix verifiable computing consortium, it is not possible for the compute nodes to collude with the other platforms.

11 Payment Handler

When the computation is successful, the compute nodes have to wait for $\mathcal{C}_t$ time units before receiving the payment. If the user does not challenge the result within $\mathcal{C}_t$, then the payment handler pay the fee to the fabrics. The fabric leader distributes the payment to the compute nodes.

12 Reputation Management

The verifiable computation framework enables the users to submit computation tasks, and the tasks are executed on compute nodes in return for an incentive. For the system to become reliable and sustainable, all the parties must act honestly. To identify and penalize malicious actors, a reputation mechanism is necessary.

The compute nodes start with zero reputation. When the nodes perform computation, it is rated based on the outcome of the computation and the user's ratings. Each node gets additional rating points for their liveness, i.e., the nodes that are active during the sporadic resource testing would get special rating points.

The requirements for a reputation system is listed below from work by Vavilis et al. [30].

- Ratings should discriminate user behavior
- Reputation should discriminate user behavior

- The reputation system should be able to discriminate "incorrect" ratings
- Reputation should be assessed using a sufficient amount of information
- Reputation should capture the evolution of user behavior
- Users should not gain the advantage of their new status
- New users should not be penalized for their status

12.1 Problem of Self-Rating

The owner of the compute node may also repeatedly submit a lightweight computation task to self-rate his or her compute node. The origin of rating and the task submission may reveal the identity the user, but it is not a straightforward process as the user may mount Sybil attacks.

12.2 Differentiation between New Users and Bad Users

When a new compute node joins the system, it does not have any reputation. A moderate reputation score can be assigned to the new nodes to differentiate them from bad nodes as a low reputation score for a new code may prevent it from gaining reputation. But, this may allow the bad actors to leave the system and join again as a new node. Therefore, it is crucial to enable the new nodes to build a reputation gradually. A joining fee along with an escrow deposit may prevent the bad nodes from rejoining the system.

12.2.1 PoW for the New User

When a new user joins the system, the joining process should not only involve resource testing, but it may also involve a PoW for the user to deter the user from mounting Sybil attacks. The simpler the joining process, the higher the chances of the Sybil as the nodes may join and leave at will to delete the poor rating. A PoW computation in combination with an escrow for the new user would reduce the likelihood of Sybil attacks. Besides, the voting power (i.e., weight) should be proportional to the age of the user in the system.

13 Summary

The key building blocks of the verifiable computation platform includes the joining process for the compute nodes, fabric management, scheduling of compute tasks, verification of the results, and the reputation management. The platform is currently under development following the approaches described in the report. During the

implementation, the approach will be fine-tuned for performance, scalability, and security.

References

[1] K. Hwang and M. Chen, *Big-data analytics for cloud, IoT and cognitive computing*. John Wiley & Sons, 2017.

[2] A. Munir, P. Kansakar, and S. U. Khan, "Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 74–82, 2017.

[3] M. Mohammadi and A. Al-Fuqaha, "Enabling cognitive smart cities using big data and machine learning: Approaches and challenges," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 94–101, 2018.

[4] Y. He, F. R. Yu, N. Zhao, V. C. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 31–37, 2017.

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: An experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002. [Online]. Available: http://doi.acm.org/10.1145/581571.581573

[6] S. Zhao, V. Lo, and C. G. Dickey, "Result verification and trust-based scheduling in peer-to-peer grids," in *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, Aug 2005, pp. 31–38.

[7] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[8] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '11. New York, NY, USA: ACM, 2011, pp. 113–124. [Online]. Available: http://doi.acm.org/10.1145/2046660.2046682

[9] A. C. Yao, "Protocols for secure computations," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 1982, pp. 160–164.

[10] S. Goldwasser, "Multi party computations: Past and present," in *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '97. New York, NY, USA: ACM, 1997, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/259380.259405

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[12] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 781–796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 90–108.

[14] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 626–645.

[15] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," *url: https://people. cs. uchicago. edu/teutsch/papers/truebit pdf*, 2017.

[16] K. Iwasaki, "Perlin: Scalable dag-based distributed ledger protocol using avalanche consensus," Perlin, Tech. Rep., 2018.

[17] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," Team Rocket, Tech. Rep., 2018.

[18] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *arXiv preprint arXiv:1804.05141*, 2018.

[19] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 619–636.

[Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[20] I. Bentov, P. Hubácek, T. Moran, and A. Nadler, "Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies."

[21] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," Tech. Rep., 1996.

[22] A. Biryukov and L. Perrin, "Symmetrically and Asymmetrically Hard Cryptography," in *Asiacrypt 2017 - Advances in Cryptology*, ser. LNCS - Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10626. Hong Kong, China: Springer, Dec. 2017, pp. 417–445. [Online]. Available: https://hal.inria.fr/hal-01650044

[23] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, "Bitcoin mining pools: A cooperative game theoretic analysis."

[24] O. Regev and N. Nisan, "The popcorn market—an online market for computational resources," in *Proceedings of the First International Conference on Information and Computation Economies*, ser. ICE '98. New York, NY, USA: ACM, 1998, pp. 148–157. [Online]. Available: http://doi.acm.org/10.1145/288994.289027

[25] S. Bhattacharya, K. Kar, J. H. Chow, and A. Gupta, "Extended second price auctions for plug-in electric vehicle (pev) charging in smart distribution grids," in *2014 American Control Conference*, June 2014, pp. 908–913.

[26] N. C. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal auction for edge computing resource management in mobile blockchain networks: A deep learning approach," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[27] P. Dütting, Z. Feng, H. Narasimhan, and D. C. Parkes, "Optimal auctions through deep learning," *CoRR*, vol. abs/1706.03459, 2017. [Online]. Available: http://arxiv.org/abs/1706.03459

[28] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," *CoRR*, vol. abs/1708.01171, 2017. [Online]. Available: http://arxiv.org/abs/1708.01171

[29] E. Staab and T. Engel, "Collusion detection for grid computing," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC,

USA: IEEE Computer Society, 2009, pp. 412–419. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.12

[30] S. Vavilis, M. Petkovic, and N. Zannone, "A reference model for reputation systems," *Decision Support Systems*, vol. 61, pp. 147–154, 2014.

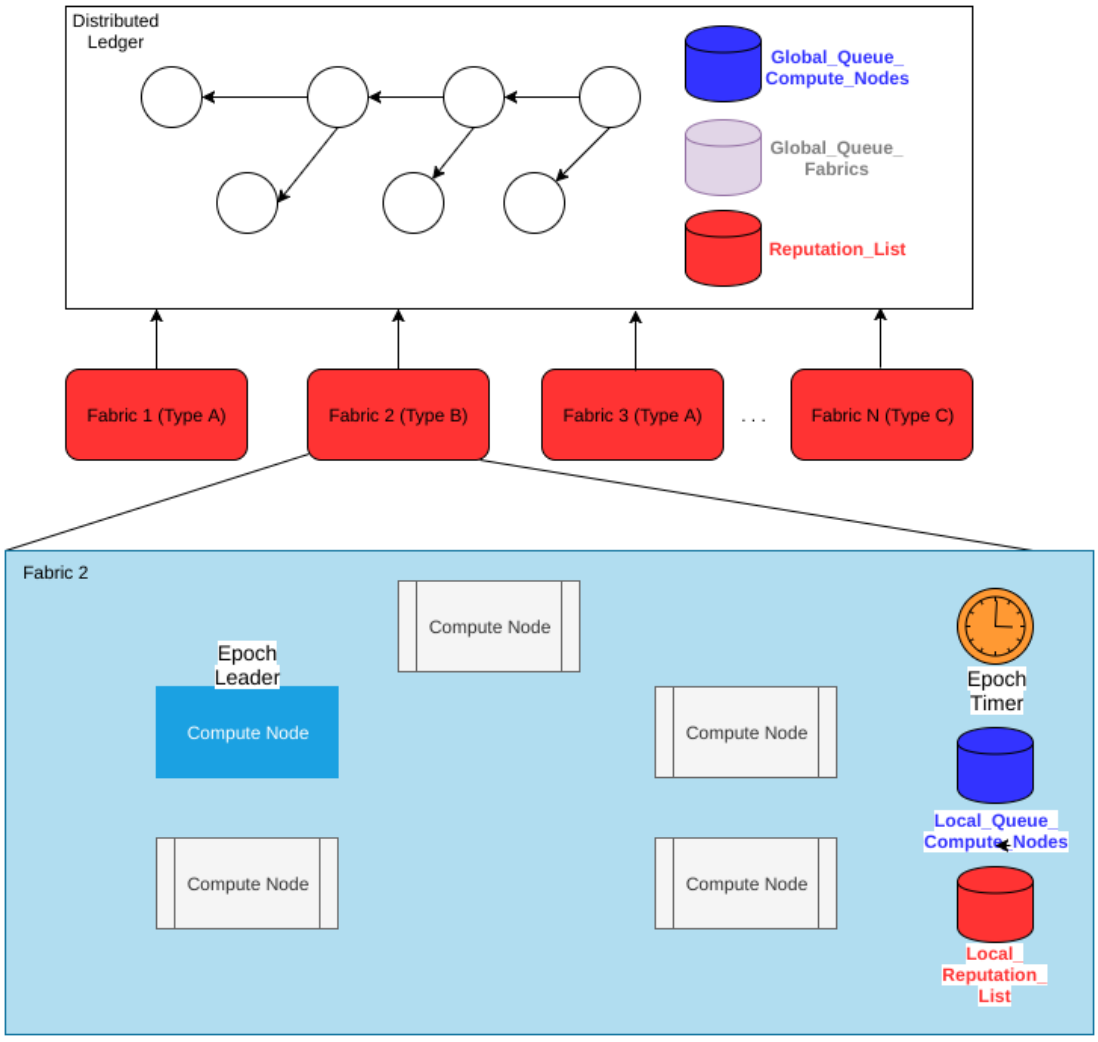

Figure 3: Fabric Architecture.

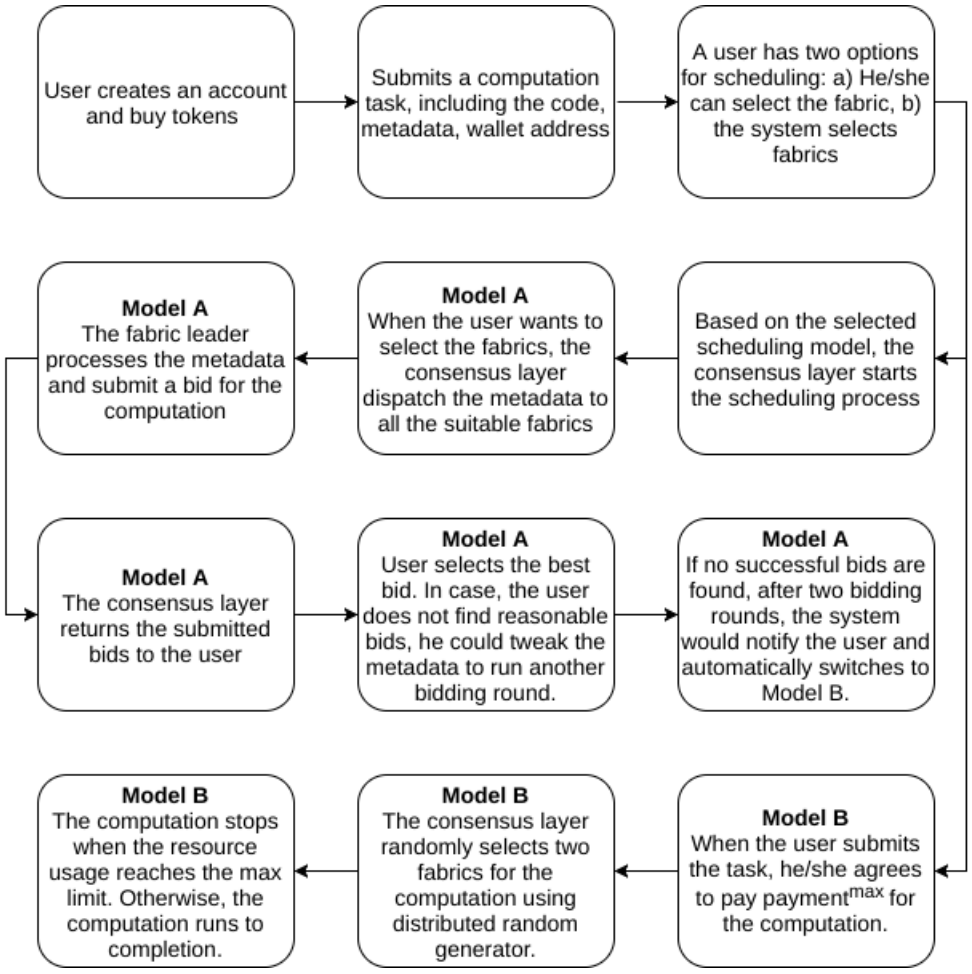

Figure 4: The Joining and Scheduling Process.