CloudAgora: Democratizing the Cloud

Katerina Doka, Tasos Bakogiannis, Ioannis Mytilinis, and Georgios Goumas

Computing Systems Laboratory,
National Technical University of Athens, Greece
{katerina,abk,gmytil,goumas}@cslab.ece.ntua.gr

Abstract. In this paper we present CloudAgora, a platform that enables the realization of a democratic and fully decentralized cloud computing market where participating parties enjoy significant advantages: On one hand, cloud consumers have access to low-cost storage and computation without having to blindly trust any central authority. On the other hand, any individual or company, big or small, can potentially serve as cloud provider. Idle resources, be it CPU or disk space, are monetized and offered in competitive fees, regulated by the law of supply and demand. In the heart of the platform lies the blockchain technology, which is used to record commitment policies, publicly verify off-chain services and trigger automatic micropayments. Our prototype is built on top of the Ethereum blockchain and is provided as an open source project.

1 Introduction

The advent of cloud computing has revolutionized the IT sector worldwide, by allowing organizations and individuals alike to opt for remote resources - be it storage, computation or applications - instead of costly and hard-to-maintain local infrastructure. In the last years, cloud computing has indeed prevailed over traditional on-premise environments as a means of executing applications and/or offering services for a wealth of reasons, including reduced costs, seemingly infinite resources purchased in a pay-as-you-go manner, scalability, ease of maintenance, etc., [8]. This fact has highly impacted a multitude of industry domains in the way they do business and has fundamentally transformed our everyday lives in the way we work and communicate [5].

Cloud services are mainly based on (a handful of) large providers that act as trusted entities for the transfer, storage and processing of user or company data. Thus, despite their reliance on fundamental principles of distributed computing, they fail to achieve full decentralization. The main disadvantages of this cloud computing model are summarized in the following:

– It carries the intrinsic weaknesses of any model based on trust: Users take for granted that providers act in the interest of their customers rather than opportunistically.
– The leading Cloud providers have invested huge amounts of money to build massive server farms and consume enormous amounts of energy for running and cooling them. Although they can provide prices that render infrastructure renting more appealing than on-premise infrastructure operation

in the majority of cases, pricing could be even more affordable, had there been a greater competition [21]. Thus, the public cloud market has become a functional monopoly where a few providers define the prices, which are non-negotiable and can be prohibitively high for applications demanding specialized hardware.
- Sovereignty over data and control over computations performed on top of them are surrendered to the big players, who thus accumulate knowledge, gaining significant competitive advantage and strengthening their already privileged position.

The missing traits, i.e., full decentralization, strong guarantees for security and integrity based on proof rather than trust and transparency in any user-provider interaction, are the ones that blockchain technology can provide. Blockchain started off as the driving force behind Bitcoin - a distributed ledger where transactions are ordered, validated and, once recorded, immutable. With the addition of smart contracts - pieces of code which are executed automatically, in a distributed manner - it quickly rose as one of the most groundbreaking modern technologies, offering a new approach to decentralized applications and disrupting a wide range of fields such as finance, IoT, insurance, voting etc., [18].

However, blockchain technology is not a panacea. Especially when it comes to storage and power devouring applications, blockchains fall short of their requirements due to the limited computing and storage capacity they offer. Typically, the most prevalent blockchains, such as Ethereum, support blocks of at most a few Megabytes, achieve a throughput of a few tens of transactions per second and accommodate a limited number of operations per smart contract [9]. As such, blockchains and smart contracts cannot be adopted as storage providers or computing engines per se, but rather as an enabling technology which keeps track of and validates off-chain operations. The challenge in this scenario is to find a secure way to guarantee the correctness of the off-chain service through the use of a publicly verifiable proof [12].

To that end we present CloudAgora, a truly decentralized cloud that allows for on-demand and low-cost access to storage and computing infrastructures. The goal of CloudAgora is to create a blockchain-based platform where participants can act either as providers, offering idle CPU and available storage, or as consumers, renting the offered resources and creating ad-hoc virtual cloud infrastructures. Storage and processing capacities are monetized and their prices are governed by the laws of supply and demand. Thus, CloudAgora democratizes the cloud computing market, allowing potential resource providers - ranging from individuals to well established companies in the field - to compete with each other in a fair manner, maintaining their existing physical infrastructure.

In a nutshell, CloudAgora offers users the ability to express a request for storage or computation and take bids from any potential provider in an auction-style manner. Anyone who can supply storage and/or computing power can become a CloudAgora provider, ranging from individuals or companies offering idle or under-utilized resources to large datacenters, traditionally operating in the Cloud market. Customers are automatically matched to resource providers according to the height of their bid and their reputation. The agreement between providers and consumers is encoded as a smart contract, which allows for traceability of actions and automatic triggering of payments. While storage and processing is performed off-chain, the integrity and availability of stored data as well as the correctness of the outsourced computation are safeguarded through proper verification processes that take place on the chain. Special consideration has been dedicated to offering the necessary incentives for a fair game, both from the provider as well as the customer perspective.

Such a solution offers significant advantages compared to the traditional, datacenter-only based cloud computing model: Providers can exploit existing idle resources for profit while consumers enjoy lower fees due to competition without having to blindly trust any central authority or big company. Through the use of blockchain technology, all services performed in the cloud are recorded and payments are automated accordingly.

Approaches similar to CloudAgora in the competitive landscape either focus exclusively on data hosting [22, 2, 20] or target specific applications, such as 3D rendering [3]. Other solutions that address secure, off-chain computation have limited applicability due to the restrictions they pose on the type of computation supported [23]. Many of these projects rely on their proper blockchain and native coins, complicating the redemption of rewards. Contrarily, CloudAgora is a fully open-source, based on Ethereum platform, which provisions both storage and computation resources.

In this paper we make the following contributions:

- We propose an open market platform, where users can trade storage and computation resources without relying on any central authority or third party. By enabling any user to become a potential resource provider, our work breaks the monopoly of the few and creates of a truly democratic and self-regulated cloud market.
- We offer a solution that addresses the provision of both storage and compute resources in a unified manner based on smart contracts. The proper publicly verifiable proofs that the off-chain service, either data or computation related, was correctly completed have been identified and incorporated to our platform.
- We implement the proposed platform on top of the most prevalent smart contract blockchain, Ethereum and provide it to the community as an open source project.

2 Architecture Overview

CloudAgora is a system that provides the basic primitives and tools for enabling a truly decentralized cloud infrastructure. Anyone that joins CloudAgora can act either as a cloud user, a cloud provider or both. By taking advantage of blockchain technology, we establish an environment where rational participants do not diverge from their expected behavior, monopoly effects are eliminated and prices dynamically adjust according to market rules. Henceforth, we refer to

CloudAgora users that provide resources as *service providers* and to users that consume resources as *clients*.

As we consider that the adoption of a system highly depends on the ease of installation and use, we propose a lightweight design that operates on top of any blockchain technology that supports smart contracts. Although our approach to the design of the system is blockchain-agnostic, we base our prototype implementation on Ethereum, one of the most popular and advanced smart contract platforms, while we keep its internals intact. This way, the whole cloud environment can run as a common application in every public or private Ethereum blockchain.

The system is hierarchically structured in two layers, namely the *market layer*, and the *storage/compute layer*. At the highest level, there is the market layer. This is an abstraction of the way the economy of CloudAgora works. This layer comprises a set of algorithms that define participants' incentives and mechanisms for the regulation of prices. The creation of a new cloud job, the decision on price levels and the assignment to a specific provider all belong to the market layer. The CloudAgora market rules are enforced through a set of smart contracts that work on-chain.

At the bottom layer, actual cloud services are provided: data persistence and computations take place. Furthermore, this layer contains algorithms that can work both on- and off-chain and ensure the provably proper operation of the whole system. The contracts of this layer audit clients and providers and guarantee that none is making profit against the rules of the market. In the following sections, we describe in more detail the two layers of our system. Section 1 presents the market layer, Section 4 demonstrates our approach to decentralized storage and Section 5 shows how CloudAgora can provide provably correct computations in a decentralized cloud environment.

3 The Market Layer

In a typical cloud scenario nowadays, a user willing to consume resources will have to choose among a few known providers (e.g., Amazon, Google, Microsoft), accept the prices they offer without the right to negotiate it and finally deploy her job. The deficiencies of this approach are twofold: (i) as only a few cloud providers determine price levels, cloud deployments in many cases end up too costly to afford and (ii) large companies accumulate vast amounts of data and get a great head start in races like the ones of machine learning and big data processing.

CloudAgora remedies these drawbacks by enabling a free market where each player can participate on equal terms. Moreover, prices are not fixed but are determined through an auction game. Since potentially anyone can be a service provider, data does not end up in the possession of a few powerful players but are expected to be distributed among all members of the system.

Let us assume a client that needs resources for either storing a dataset D or computing a task T. The client broadcasts a description of D or T and initiates an

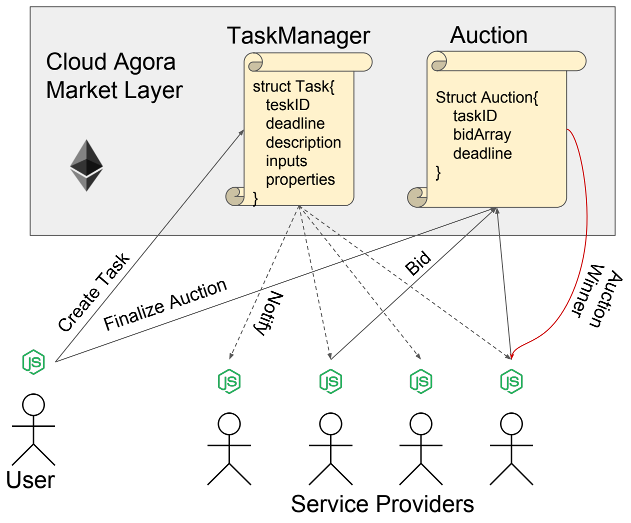

Fig. 1: Market Layer

auction game. Based on this description and the assessed difficulty/cost, anyone interested in providing resources can make an offer. The client finally selects the provider with the most appealing offer (in terms of both price and credibility) and assigns her the job. For guaranteeing integrity and transparency, the market layer is implemented as a set of smart contracts that operate on-chain.

Figure 1 illustrates the workflow followed in a CloudAgora auction. For supporting the required functionality for both clients and service providers, we have implemented the corresponding NodeJS clients. Each CloudAgora client is a Dapp that exposes a specific API and can interact with the blockchain.

Initially, the client interacts with the *TaskManager* contract and creates a new task (storage or computational). Each task comprises a structure that contains a set of mandatory and a set of optional fields. The mandatory fields of a task are: (i) its unique id, (ii) an expiry date, until when the service provider commits to deliver resources and (iii) a description indicative of the tasks's difficulty. For storage tasks, this description can be the size of the dataset the client needs to store and for computational tasks, the amount of required gas if user code is converted to Ethereum Virtual Machine (EVM) assembly. The optional fields may contain task-specific information (e.g., the schema of a dataset, an application parameter, etc.).

As soon as the task is created, the TaskManager contract calls the *Auction* contract and creates a new auction. Every auction carries three pieces of information: (i) the corresponding task id, (ii) bid array: a structure where all bids are maintained and (iii) a deadline. If the deadline expires and a winner has not been found, the Auction contract cancels both the auction and the task. Auction deadlines are measured in chain blocks. When both the task and the auction are ready, a *taskCreation* event is emitted on the blockchain and all interested parties, i.e., service providers, listen to it. Any service provider that is interested in getting paid for the specific task places her bid at the Auction contract. Upon the receipt of a new bid, the contract checks if the received bid is better than the best it currently maintains in its structure. If not, the bid is discarded. Otherwise, the bid array is updated and a *newBid* event is emitted to both the client and the service providers. The client can inspect the current bid and if it suits her, she can finalize the auction and select provider. A service provider can also inspect the last submitted bid and evaluate if she is willing to make a new offer or not.

We mentioned that only offers better than the current best are inserted into the bid array. A question that naturally arises is what is the criterion for comparing bids. A naive approach would suggest to always choose the provider that offers the lowest price. However, this would encourage malicious players to offer services in extremely low prices. For tackling this problem, in CloudAgora, we employ two distinct mechanisms. When the auction is finalized, the selected provider has to put a collateral until the corresponding task expires. If the provider fails to deliver, the collateral is never returned to her but instead is handed to the client as a refund. The size of the collateral is automatically set to a value greater than the total payoff that the provider will receive if she successfully delivers the task. This way, the provider is incentivized to play by the rules.

Along with the collateral, we also establish a reputation-based system. Providers with a bad reputation should be penalized even when they offer appealing prices and providers that are renowned for their quality of work should have the right to claim higher prices. Thus, for comparing bids we use a function $f\left(price, reputation\right) \in \Re$ in order to meet both criteria. The bid that wins the auction is the one with the highest f-value.

However, it is still unclear how reputation scores are computed and assigned. Upon the completion of a task (be it storage or computational) the client calls the *finalize* function of the TaskManager contract. This function takes as input a proof of the task's success and a binary reputation score: 1 denotes success and 0 states that the provider failed to deliver. As the proof of success is sent along with the reputation score, the miners that execute the *finalize* function can verify if reputation is correctly assigned and prevent an *unfair ratings attack* [10]. The aggregate reputation of a provider is calculated by summing up all reputation scores that have ever been assigned to her. It is this aggregate reputation that is used by the f-function in order to compare bids.

4 Storage

In this section we highlight challenges we met and describe our approach to implementing decentralized storage over blockchain in CloudAgora. First, we describe a number of desired properties we consider for a remote data storage system. Then we elaborate on how we provide guarantees for those properties on a trustless environment. Finally, we outline a typical storage workflow in CloudAgora.

4.1 Challenges Specific to Storage

When we consider remote data storage we often come across a wide variety of client requirements. However, all seem to be related to a small number of properties we want a remote data storage system to have. Specifically, in the evaluation of such a system we mostly consider *Data Integrity*, *Data Availability*, *Data Recovery* and *Privacy*. That is the ability to retain our data intact, to access our data at any time, to be able to address data loss and to keep others from accessing our data.

Moving from the more established cloud storage model to a decentralized storage model over blockchain requires us to reconsider our approach regarding the above properties. Existing cloud storage systems are based on trust, reputation and SLAs, however these mechanisms are not directly transferable to a trustless environment such as a blockchain.

At this point we have to note that the blockchain itself can guarantee those properties. However, as we mentioned before, it is not designed to be used as a decentralized storage layer. The cost of storing large data volumes on-chain is prohibitive and its use as a decentralized storage system is impractical since all the data have to be replicated in all the peers. As a result, we have to develop off-chain solutions that can guarantee *data integrity, availability, recovery and privacy* using the blockchain only for bookkeeping, that is only as a distributed ledger that cannot be tampered.

4.2 Our Approach

Given CloudAgora should operate on a trustless environment we have to establish ways to enforce the desired behavior for each one of the participants. Since we base CloudAgora on a open, permission-less blockchain we observe that we cannot enforce any particular behavior. That is because the participants are free to join or leave the process at any time, or even continue with a different identity. As a result we restrict our hypothesis and assume that the participants are rational players that do not exhibit altruistic behaviors. This assumption allows us to use incentives to guide the participants behavior and provide some guarantees related to the desired properties of a remote data storage system. In practice we use a combination of incentives and cryptographic tools to ensure those properties.

In a typical remote data storage scenario we would have a *client* who wants to store data remotely and one or more *providers* who offer data storage. From the *market layer* we have a way to choose one *provider* for a *client*. Therefore, from now on we assume that we have two parties a *client* and a *provider*. They can interact either on- or off-chain, their on-chain interactions are governed by a smart contract. We call this smart contract the *storage contract*, it is a contract between the two parties for storing specific data on the *provider's* side. It has an end date provided by the *client* and the *provider's* payment as it is defined by the *market layer*. The payment amount is transfered from the *client* to the *storage contract* and therefore it is managed by it. Given this configuration, in the following Sections, we describe how we approach and guarantee each one of the properties mentioned.

Data Integrity: To ensure that the data of the *client* will not be tampered while at rest we use two different mechanisms: incentives and Merkle trees.

We first incetivize the *provider* to guarantee the integrity of data by requiring from her to provide collateral in case of data tampering. In order for the *provider* to accept the *storage contract* she has to transfer to the contract a previously agreed upon amount. As a result if the *provider* cannot prove to the *storage contract* that the data have not been tampered, she will lose the collateral.

We further icentivize the *provider* through our reputation system used in the *market layer*. Having in place a reputation system that aids *providers* with good reputation to get better storage deals, we can penalize the reputation of a *provider* that fails to guarantee data integrity. This penalty will influence its future deals and therefore as a rational player she is encouraged to provide good data integrity guarantees.

The mechanism we use to enable the *provider* to prove that she owns the *client's* data is based on Merkle trees [16]. The *client* calculates the Merkle tree of the data to be stored and saves its root hash in the *storage contract.* In the same manner the *provider* verifies the root hash before accepting the contract. As a result, at any point in time the *provider* can send a number of Merkle proofs [16] to the *storage contract.* The contract by verifying that the given proofs match the root hash of the data's Merkle tree, ensures that the *provider* still owns at least part of the original data. Increasing the number of proofs required, increases the probability that the original data remain intact.

Data Availability: To deal with availability we require the *provider* to reply within a certain time frame to random challenges initiated by the *client.* Those challenges involve the *client* asking for a Merkle proof of a specific part of the data. The *client* can perform the challenge either on- or off-chain. In the first case it is the *storage contract* that verifies the proof in the second it is the *client.* If the check is performed on-chain and the *provider* fails to reply within the specified time frame the contract is invalidated and she loses the collateral as well as reputation. That way the *provider* is incentivized to have a certain response time in regard to data availability. Having the response time frame specified as part of the contract gives the *provider* a way to implement hot or cold storage options.

Data Recovery: The incentives provided for *Data Availability and Integrity* could be considered sufficient to cover the case of data loss as well. In such a case the *provider* would not be able to prove that she owned the original data and eventually lose its collateral and reputation. However, given that the Merkle proofs mechanism can only guarantee the availability of a percentage of the original data, we incorporate erasure codes as a way of lowering even further the possibility of data loss or corruption. In a nutshell, erasure coding expands and encodes a dataset with redundant data pieces and breaks it into n fragments in a way such that the original dataset can be recovered from a subset of the n fragments. There is a large collection of erasure codes available today, we opted for the Reed-Solomon codes [17] because of their popularity and wide use. By erasure encoding the data before sending them to the *provider*, the *client* ensures that retrieving only a part of the stored data is enough to restore all of the original data. This process is transparent to the *provider* and the *storage contract*, that is both the contract and the *provider* treat the *client's* data the same way whether they are erasure encoded or not.

Privacy: The simplest way to handle privacy is at the *client* side by encrypting the data before transmission. As with erasure codes, the *provider* and *storage contract* handle the *client's* data identically whether they are encrypted or not. That gives the *client* the freedom of choosing any encryption algorithm or even not using encryption if not required.

Thus, by assuming rational players and through the use of monetary and reputation based incentives, we are able to guarantee *Data Integrity, Availability and Recovery.* This is enabled by creating Merkle trees on the *client's* data and using erasure codes to lower the probability of data loss.

4.3 Storage Workflow

In this section we describe in detail the remote data storage workflow as well as the life-cycle of a *storage contract* and its state transitions.

Storage Workflow: At this point we assume that the *client* has already interacted with the *market layer* and the corresponding auction is finalized. As a result a *storage contract* is created that binds a *client* with a *provider*. The contract contains the addresses of the *client* and the *provider*, the root hash of the Merkle tree of the *client's* data, the end date of the contract as well as the payment and collateral amounts. Those amounts are transfered to the contract upon its creation and therefore managed by the contract logic. We note that any preprocessing to the data to be stored should be performed by the *client* before the initial auction phase. Typically the data preprocessing includes encryption and erasure encoding as mentioned in the previous section. Given that in the *storage contract* we store the root hash of the Merkle tree of the data we must perform any preprocessing before the Merkle tree calculation.

As a second step, the *client* sets up a server to serve the data. The *provider* downloads the data, computes their Merkle tree and verifies their integrity by matching the root hash of the tree with the one stored in the *storage contract*. If the hashes match, the provider activates the contract. From that point on, the *client* can safely assume that the data are stored remotely.

After the end date of the contract, the *provider* can collect the payment. To do so she has to prove that she still has the data at its possession, he does so by sending a number of Merkle proofs to the *storage contract*, if the proofs are valid the contract releases the funds and transfers them to the *provider's* address. If the *provider* cannot prove that she possesses the *client's* data the contract transfers all the funds to the *client's* address since it assumes that the *provider* lost the data and therefore the *client* should receive the collateral.

At any point in time after the contract activation and before the end date of the contract, the *client* can request its data from the *provider*, this operation is performed off-chain. Although, it is not possible for the *client* to request the data on-chain, she can achieve the same goal by asking a sufficient number of Merkle proofs and restoring its original data from the proofs. This second alternative, however, is not practical but the system's incentives are against it, since providing a Merkle proof to the *storage contract* costs to the *provider* gas.

Additionally, while the contract is active the *client* can challenge the *provider* by requesting a Merkle proof for a specific data block. This process can be performed either on- or off-chain and is used as a safeguard against data tampering.

In the case of an off-chain challenge, we have to note that for the *client* to be able to challenge the *provider* she would have to decide beforehand on a number of challenges and store that number of data blocks locally before serving the data to the *provider*. This is because at the time of the challenge in addition to the Merkle proof the data block that matches that proof is also verified. Thus, when the *client* challenges the *provider* to prove that she owns a specific data block D, the *provider* sends back to the client the given data block D as well as the path of the Merkle tree from D to the root of the tree. At that point, the *client* can verify that the *provider* has D at its possession as well as that D is part of the original data.

In the case of an on-chain challenge, the *client* requires from the *provider* to prove that she owns a number of different data blocks. As a result the *provider* has to send to the *storage contract* a given number of randomly selected data blocks as well as the corresponding Merkle tree proofs. At that point the *storage contract* produces the hash of each data block and combines it with the corresponding proof verifying if the block leads to the Merkle tree root that is stored on the contract. By requesting a given number of Merkle proofs at each challenge the *client* can ensure with good probability that the *provider* has the original data at its possession.

Storage Contract Life-Cycle: The life-cycle of a *storage contract* can be represented by its state transitions as depicted in Fig. 2. The dashed transitions represent actions performed by the *provider* while the solid ones by the *client*.

Transitions t_2, t_4 and t_7 are triggered by the *provider*. In t_2 the *provider* receives the data and proves that she owns it in order to activate the contract. In t_4 the *provider* proves on-chain that she still has possession of the data while in t_7 performs the same proof in order to complete the contract and receive the payment.

Transitions t_0, t_1, t_3, t_5 and t_6 respectively are triggered by a *client* action. In t_0 the *client* through his interaction with the *market layer* creates a new *storage*

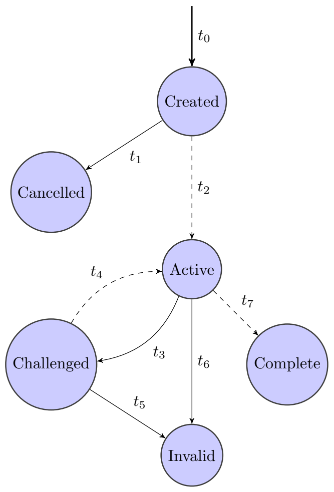

Fig. 2: Storage Contract State Diagram

contract and specifies payment amount, collateral, provider and contract end date. In t_1 the *client* has the option to cancel the contract before it is accepted by the *provider*. In t_3 the *client* challenges the *provider* by requesting an on-chain proof that she still owns the data. t_5 can be triggered by the *client* in the case the *provider* failed to prove that she owns the data, in that case the contract is invalidated and the collateral is transfered to the *client*. Finally, in t_6 we have the same transition with t_5 only after the contract end date.

5 Compute

Providing secure computations by untrusted parties presents its own challenges. For the following discussion, we make two observations:

a) In CloudAgora there exists a trusted network (blockchain) that correctly performs small computational tasks. While any algorithm can be developed as a smart contract and executed on the blockchain, we particularly focus on small tasks. Heavy contracts lead miners to the notorious *Verifier's Dilemma*. On-chain verifications that require non-trivial computational efforts will fail to execute correctly in rational miners and the whole chain will be vulnerable to serious attacks [14].

b) Participants are rational in the sense that they act to maximize individual profits. A CloudAgora member is eager to solve or verify a task only if she expects to have a monetary profit.

Based on these observations we develop a truebit-like [19] game, where outsourced algorithms are executed off-chain and blockchain is only used for correctness proofs. The outline of such a game consists of the following steps:

- The user announces a task and a task description is placed on-chain.

– Depending on the protocol, a solver is selected for executing the task. In the traditional truebit game, the solver is selected at random. In CloudAgora, a solver/provider is selected based on the auction described in Section 3.
– The solver privately performs computations off-chain and only after completion, she reveals the solution on the blockchain.
– While solver is computing the task, any other member of the system can also compute it in private and act as a verifier. If a verifier agrees with the solution provided by the solver, the solver gets paid and the game stops. In case of a disagreement, the verifier can challenge the solver and an interactive proof takes place on-chain. If the solver proves to be malicious, the verifier receives solver's reward and solver looses the deposited collateral we discussed in Section 3. If the verifier has triggered a false alarm, she is obliged to pay for the resources wasted due to the interactive game.

5.1 Interactive proof

In this Section we discuss how disputes are resolved in case of disagreement between a solver and a verifier. First of all, we must be sure that both parties compute exactly the same program and that the architecture of the infrastructure that each party has employed does not affect the result. For this reason, an announced task is first converted to an assembly-like intermediate representation. Then, both parties privately compile a tableau of Turing Machine (TM) configurations, where each time step of the task is mapped to its complete internal representation (tape contents, head position, machine state). The game also determines a parameter c that declares how many configurations the solver broadcasts to the blockchain in each roud and a timeout period within which verifiers and the solver must respond. Failing to do so leads to immediate loss for the non-responding party.

The main loop of the game goes as follows:

– The solver selects c configurations equally spaced in time across the current range of dispute. She then computes c Merkle trees of the Turing tableau where each tree corresponds to the $\frac{1}{c}$ of the current range of dispute. Each leaf of these trees is the complete TM state for a specific time step. The roots of all these Merkle trees are placed on the blockchain.
– The verifier responds on-chain with a number $i \leq c$, indicating the first time step in this list that differs from her own.
– The process continues recursively, considering as dispute range the one between the $(i-1)$-st and i-th indexed configurations.

After some rounds, the game converges to the first disputed computational step t. The solver then provides paths from the Merkle tree root to its leaves for the moments $t-1$ and t. The transition of the TM state at time $t-1$ to the one at time t is computed on-chain and the disagreement is resolved by the miners.

6 Prototype Implementation

We implemented a prototype of CloudAgora as a dApp over Ethereum. As a result the contracts that implement the auction and govern the on-chain *client, provider* interactions are in Solidity. All the off-chain logic is implemented in NodeJS using web3.js to interface with the Ethereum blockchain. We used parts of the truebit codebase to implement the compute module of CloudAgora while the Storage is implemented from scratch.

CloudAgora introduces a non-negligible performance overhead to an application execution compared to executing the application over the same hardware without using CloudAgora. This overhead includes the latency of smart contract operations, the latency of commiting transactions to the blockchain as well as the overhead attributed to the truebit protocol. Thus, the blockchain of choice heavily determines the performance overhead. Since our prototype relies on Ethereum, CloudAgora inherits its performance characteristics [11].

7 Related Work

CloudAgora is a platform that facilitates the provision of storage and computation resources in a fully distributed and democratic manner, using the Ethereum blockchain to record commitment policies, publicly verify them and automate micropayments. Related work includes blockchain-based projects that offer off-chain computations, data hosting and processing services.

In the domain of data hosting, projects such as Storj, Filecoin and Sia permit users to rent unused storage space, using a blockchain to guarantee the correctness of the service offered. Sia [20] supports smart contracts between storage suppliers and users, which are stored and executed in its proper blockchain. Integrity and existence of a piece of data on a remote host is guaranteed using Merkle proofs and associated final payment is enforced automatically through the contract. Storj [22] is based on Kademlia Distributed Hash Table (DHT) [15] to offer a P2P cloud storage network that builds on top of any smart contract blockchain. Erasure coding is employed as a redundancy mechanism, while audits are based on Merkle proofs. Filecoin [2] relies on zk-SNARK [7], a cryptographic tool for zero-knowledge verifiable computation, to provide what the creators call *Proof-of-SpaceTime*, i.e., evidence that some data has been stored throughout a period of time. This is made possible through iterations of challenges and responses. All the above solutions exclusively address the case of storage provisioning, while CloudAgora provides both storage and computational resources.

Projects such as GridCoin, Enigma, Golem, Dfinity and iExec offer both distributed storage and computation services.

The GridCoin [4] project creates a cryptocurrency as a reward for computations provided to BOINC-based volunteer projects for scientific purposes. It utilizes its own consensus protocol, called Proof-of-Research, which replaces the traditional Proof-of-Work puzzle with useful work. Since it purely concerns volunteer grid computing projects, it is mainly limited to altruistic sharing of resources for scientific research.

Enigma [23] utilizes Multi-Party Computation (MPC) [13] protocols to ensure correct execution while preserving data privacy. Due to the fact that it heavily relies on homomorphic encryption to allow nodes to operate on encrypted shards of data, it poses restrictions regarding the type of computation it can support, thus limiting its applicability and hindering its wide adoption in practice.

Golem [3] is built on top of the Ethereum blockchain. It mainly offers software services and thus focuses on specific computation tasks (e.g., 3D rendering). Proof of correct execution is available through Truebit style challenges, where user reputation is not (yet) taken into account. Contrarily, CloudAgora aims to support any type of task. Moreover, although CloudAgora too employs Truebit as a means of verifying the correctness of the outsourced computation, the reputation mechanism and the auction game it adopts for matching resource requests to providers enhances Truebit's mechanism of allocating tasks to Solvers.

Dfinity [1] is a blockchain aiming to act as a replacement to smart contract platforms like Ethereum by creating a decentralized cloud computer, the "Internet computer", which will host the next generation of Dapps. iExec [6] is a project where computation audits are performed through the so called Proof-of-Contribution. This proof is based on a voting scheme that takes into account user reputation and distributes rewards based on it in a weighted manner. It builds its proper blockchain with trusted nodes using a Proof-of-Stake consensus mechanism. CloudAgora is an open-source platform, based on Ethereum, which is an already popular and widely adopted blockchain.

8 Conclusions

In this paper we presented CloudAgora, a platform that allows for on-demand and low-cost access to storage and computing infrastructures. In CloudAgora participants can act either as providers, offering idle resources, or as consumers, requesting resources. Such requests are expressed as auction smart contracts, where any potential provider can place bids. The height of the offer and the credibility of the provider determines the final choice. The agreement between providers and consumers is encoded as a smart contract, which allows for traceability of actions and automatic triggering of payments. While storage and processing is performed off-chain, the integrity and availability of stored data as well as the correctness of the outsourced computation are safeguarded through proper verification processes that take place on the chain. Our prototype implementation uses Ethereum as the underlying blockchain technology.

Acknowledgments

This research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning 2014-2020" in the context of the project "Data Sovereignty through the use of Blockchain"(MIS 5004883).

References

1. dfinity: The Internet Computer. https://dfinity.org/
2. Filecoin. https://filecoin.io/
3. Golem. https://golem.network/
4. GridCoin White Paper. https://www.gridcoin.us/assets/img/whitepaper.pdf
5. How cloud computing is changing the world ... without you knowing. https://www.theguardian.com/media-network/media-network-blog/2013/sep/24/cloud-computing-changing-world-healthcare
6. iExec Blockchain-Based Decentralized Cloud Computing. https://iex.ec/
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium, USENIX Security 14). pp. 781–796 (2014)
8. Boss, G., Malladi, P., Quan, D., Legregni, L., Hall, H.: Cloud computing. IBM white paper 321, 224–231 (2007)
9. Croman, K., et al.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security. pp. 106–125. Springer (2016)
10. Dennis, R., Owen, G.: Rep on the block: A next generation reputation system based on the blockchain. In: 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST). pp. 131–138. IEEE (2015)
11. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: A framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 1085–1100. ACM (2017)
12. Eberhardt, J., Tai, S.: On or off the blockchain? insights on off-chaining computation and data. In: European Conference on Service-Oriented and Cloud Computing. pp. 3–15. Springer (2017)
13. Goldreich, O.: Secure multi-party computation. Manuscript 78 (1998)
14. Luu, L., Teutsch, J., Kulkarni, R., Saxena, P.: Demystifying incentives in the consensus computer. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 706–719. ACM (2015)
15. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: International Workshop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)
16. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings. pp. 369–378 (1987)
17. Reed, I.S., Solomon, G.: Polynomial Codes Over Certain Finite Fields. Journal of the Society for Industrial and Applied Mathematics pp. 300–304 (1960)
18. Swan, M.: Blockchain: Blueprint for a new economy. " O'Reilly Media, Inc." (2015)
19. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains. URL: https://people. cs. uchicago. edu/teutsch/papers/truebit pdf (2017)
20. Vorick, D., Champine, L.: Sia: Simple decentralized storage. White paper available at https://sia. tech/sia. pdf (2014)
21. Wang, H., Jing, Q., He, B., Qian, Z., Zhou, L.: Distributed systems meet economics: pricing in the cloud (2010)
22. Wilkinson, S., Boshevski, T., Brandoff, J., Buterin, V.: Storj a peer-to-peer cloud storage network (2014)
23. Zyskind, G., Nathan, O., Pentland, A.: Enigma: Decentralized computation platform with guaranteed privacy. arXiv preprint arXiv:1506.03471 (2015)