computers

MDPI

Article

Leveraging Blockchain Technology to Break the Cloud Computing Market Monopoly

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

Computing Systems Laboratory, National Technical University of Athens, 15780 Athens, Greece

* Correspondence: abk@cslab.ece.ntua.gr (T.B.); gmytil@cslab.ece.ntua.gr (I.M.); katerina@cslab.ece.ntua.gr (K.D.); goumas@cslab.ece.ntua.gr (G.G.)

Received: 10 December 2019; Accepted: 6 February 2020; Published: 10 February 2020

Abstract: Cloud computing offerings traditionally originate from a handful of large and well established providers, which monopolize the market, preventing small players and individuals from having a share. As a result, the few, blindly and perforce trusted entities define the prices and manage to gain a significant competitive advantage by exploiting the knowledge derived by users' data and computations. To tackle this monopoly and empower the democratization and full decentralization of the cloud computing market, we present CloudAgora, a platform that enables any potential resource provider, ranging from individuals to large companies, to monetize idle resources competing on equal terms, and allows any cloud consumer to enjoy access to low-cost storage and computation without having to trust any central authority. The key enabler of the platform is Blockchain technology, which is used to record commitment policies through the use of smart contracts, publicly verify off-chain services, both storage and computation related, and trigger automatic micropayments. On one hand, cloud consumers have the chance to request storage or compute resources, upload data, and outsource task processing over remote, fully distributed infrastructures. Although such infrastructures cannot be a priori trusted, CloudAgora offers mechanisms to ensure the verifiable validity of the outsourced storage and computation, discourage potential providers from behaving maliciously, and incentivize participants to play fair. On the other hand, providers are able to participate in auctions, placing bids for storage or computation tasks, serve requests, and offer validity proofs upon request. Our prototype is built as a Dapp on top of Ethereum and is available as an open source project.

Keywords: cloud computing; blockchain; proofs

1. Introduction

Over the last decade, cloud computing has become a game changer for a multitude of industry domains, ranging from education and healthcare to agriculture and banking, allowing them to keep pace with the ever shifting market demands and opportunities [1,2]. Cloud computing helps businesses to optimize the resources they have at hand and reduce the cost and complexity of maintaining a local infrastructure by allowing them to rely on remote, seemingly infinite resources—be it storage, computation, or applications—in a pay-as-you-go manner. Apart from the industry, cloud computing has fundamentally transformed our everyday lives in the way we work and communicate, with storage, e-mail, and messaging services being just a few prevalent examples. The reduced costs, scalability, and ease of maintenance that cloud computing offers render it the solution of choice for organizations and individuals alike as a means of executing applications and/or offering services compared to traditional, on-premise environments [3].

Indeed, surveys show that 90% of existing companies are on the cloud, proving that cloud computing has long become mainstream [4]. The year-to-year growth rate has been astounding, with forecasts predicting that cloud data centers will process 94% of workloads in 2021 [5]. Due to

this massive adoption, the cloud computing market is blooming. Valued at $272 billion in 2018, it is expected to reach $623.3 billion by 2023 [6]. Despite its constant growth, this multi-billion dollar market remains a monopoly, with just a handful of big companies enjoying its largest share. It has been indicated that the three largest cloud providers combined accounted for 57% of the global cloud computing market in 2018 [7].

Apart from the financial implications that any monopoly has in the global market, the exclusivity of the cloud market raises issues of authority and trust: A limited number of large providers unavoidably act as trusted entities for the transfer, storage, and processing of user or company data. However, trust should not be taken for granted, since security breaches, cloud leaks, or even user data abuse (e.g., analytics performed by the providers themselves to derive knowledge about trends, patterns, and behavior) often see the light of publicity.

Thus, the traditional cloud computing model, where software, platforms, or infrastructure are offered by a single provider, fails to achieve full decentralization, despite its reliance on fundamental principles of distributed computing. The main disadvantages 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, which is not always the case.
- The leading cloud providers have invested substantial 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 [8]. 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 in the global scene.
- As data encryption on the user side is not a common practice, cloud services are vulnerable to security risks and privacy breaches that could potentially expose private data.

Blockchain technology seems like the perfect alternative, since it inherently tackles all the aforementioned issues: It offers full decentralization and strong guarantees for security and integrity based on proof rather than trust and transparency in any user-provider interaction. Blockchain has originally been introduced as a data structure consisting of a list of blocks, interlinked by cryptographic hash pointers and containing cryptocurrency transactions [9]. As such, a blockchain is a tamper-proof, distributed ledger where transactions are ordered, validated, and, once recorded, immutable, thus enabling transactions to be conducted in a secure and verifiable manner, without the need of any trusted third-party. With the addition of smart contracts—pieces of Turing-complete 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, and voting [10].

Blockchain technology, however, cannot be used as a substitute of cloud computing, as it bares significant limitations that prohibit its use as a storage or processing engine. 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 [11]. 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].

As a remedy, we present CloudAgora [13], a blockchain-based platform that allows the formation of truly decentralized clouds and enables on-demand and low-cost access to storage and computing infrastructures. The goal of CloudAgora is to provide any interested party equal chances to participate in the process of resource negotiation by acting either as a provider, offering idle CPU and available storage, or as a consumer, 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 in an open, competitive market. 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, be it centralized or distributed, 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. Through a publicly open auction process, which guarantees transparency, customers are automatically matched to the most low-priced and reputable resource providers. 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 in both central and distributed settings 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 in order to discourage malicious or selfish users who would be tempted to manipulate the system to their advantage. Moreover, fault-tolerance and service availability, both crucial aspects of any distributed system, are safeguarded through system design choices that rely on redundancy.

The proposed solution offers significant advantages compared to the traditional, datacenter-only-based cloud computing model:

- Any infrastructure owner, regardless of size or leverage, can make a profit out of existing idle resources by offering them for remote storage or computation. Thus, the multi-billion dollar cloud computing market becomes more democratic and open, offering companies and individuals a chance to enjoy a fair share.
- Consumers enjoy lower fees due to competition, since any potential provider can place bids on a specific storage or processing task through a publicly open auction process.
- Consumers do not have to blindly trust any central authority or big company. Trust is replaced by cryptographic proofs and maintained by a network of decentralized peers that come to a consensus.
- Through the use of blockchain technology, all services performed in the cloud are recorded and verified, while payments are automated accordingly upon successful completion of the undertaken tasks.

The majority of solutions related to CloudAgora either only focus on the provision of storage services [14–16] or consider specific applications, such as 3D rendering [17]. Other approaches that address the provision of secure, off-chain processing services are not generic enough to support any type of computation [18]. Moreover, a significant number of the aforementioned works build their own blockchain substrate and make use of native coins, losing thus their portability and complicating the redemption of rewards. Contrarily, CloudAgora provisions both storage and computation resources and is freely available as an open source project. Moreover, it is completely decoupled from the

underlying blockchain of choice: even though our current prototype is powered by Ethereum, it can work with any blockchain platform that supports smart contracts.

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 into our platform.
- We extend our solution to cover the cases of distributed storage and computation. To that end we describe the mechanisms adopted to render CloudAgora functional in a setting where multiple providers are involved in a specific task.
- We discuss fault-tolerance, availability, and security aspects of CloudAgora, providing an analysis of the trade-offs and best practices concerning parameter selection.
- We implement the proposed platform on top of the most prevalent, smart contract supporting blockchain, Ethereum, and provide it to the community as an open source project. Moreover, we extensively describe our implementation choices, giving adequate technical details.

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 take up the role of 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*.

The requirements that dictate the architecture of CloudAgora can be drawn from the following motivating use case scenario: A client needs to rent infrastructure and use it as a backup storage or run a demanding computational task on top of it, with the lowest possible cost. Thus, the price must be negotiable and shaped by the rules of free competition. Moreover, the client does not want to hand in the sovereignty of his/her data to any of the existing big cloud companies, but at the same time needs to ensure the credibility of the service provider. Thus, they require a comprehensive and friendly way to upload data and assign computational tasks to peer-members of a decentralized infrastructure according to the prices they offer, while data integrity and result validity control should be supported in an anytime fashion. Any individual or company able to serve a client's request should be able to make an offer, serve requests—if chosen—and offer validity proofs upon request.

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 distributed application (Dapp) in every public or private Ethereum blockchain.

The system is hierarchically structured in two layers, namely the *market layer*, and the *storage/compute layer*. The market layer constitutes the topmost level and acts as 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 are making a 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, and Microsoft), accept the prices they offer without the right to negotiate them, and finally deploy his/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 rather determined through an auction game. Since potentially anyone can be a service provider, data do 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 auction game. Based on this description and the assessed difficulty/cost of the task, anyone interested in providing resources can make an offer. The client finally selects the provider with the most competitive offer 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 each time a client submits a job to CloudAgora. Agents (client/service provider) are implemented as DApps that expose specific APIs and can interact with the blockchain. For submitting a new task, the client first has to create the corresponding auction. This is accomplished through the *AuctionFactory* contract (Step 1 in Figure 1). AuctionFactory is called by the client with three input parameters, namely, the *auction deadline*, the *task deadline*, and a score indicating the *difficulty* of the task. This score represents the file size in the case of storage tasks and the required gas if user code is converted to the Ethereum Virtual Machine (EVM) assembly in the case of computational tasks.

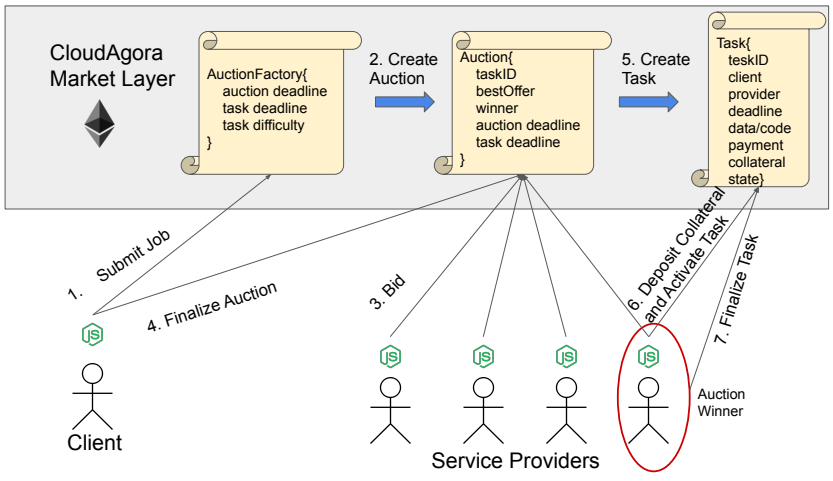

Figure 1. The market layer.

Having this information, AuctionFactory creates a new *Auction* contract for the specific task, with a unique *taskID* (Step 2 in Figure 1) and the user-provided information of the previous step (*auction deadline*, the *task deadline*, and *difficulty*). When a new auction is created, an event is broadcasted to all interested parties. Any service provider that is interested in getting paid for the specific task places her bid at the corresponding Auction contract (Step 3 in Figure 1). Upon the receipt of a new bid, the contract checks if it is better than the current best offer. If not, the bid is discarded. Otherwise, the *bestOffer* is updated and a *newBid event* is emitted to the blockchain for all interested parties, both the client and the potential service providers, to see. A service provider can inspect the new best offer and evaluate if s/he is willing to bid again or not. The client can also inspect the current winning offer. If it suits him or her, s/he can finalize the auction and select a provider (Step 4 in Figure 1). The auction is also finalized in case the deadline expires.

When an auction is finalized, a *Task* contract is automatically created (Step 5 in Figure 1). This contract lies right at the heart of CloudAgora. It binds the client and the selected service provider with an actual task description and contains the following information:

– *taskID*
– *client*
– *provider*
– *deadline*
– *collateral*
– *payment*
– *data/code*
– *state*

The first six pieces of information are known at the time of creation and are autocompleted by the Auction contract. Details are as follows:

- The *taskID* is the unique identifier of the task and is inherited by the Auction contract.
- The *client* and the *deadline* of the task are known since the very beginning of the process, upon initial user interaction with the AuctionFactory.
- The selected textitprovider and textitpayment amount comprise the result of the auction.
- The *collateral* is an amount that the service provider has to place on the Task contract as a security deposit in order to guarantee that s/he will not violate the rules of the game. By depositing the collateral, the provider accepts and activates the task (Step 6 in Figure 1). If the provider cheats, the collateral is automatically transferred to the client. Otherwise, it is returned to the service provider along with his/her payment for delivering the task. The size of collateral is a function of the decided payment and the employed function is a configurable parameter of the system. In the absence of such a guarantee, malicious players would be encouraged to offer services in extremely low prices. As proofs and validity checks take place on-chain, collateral incentivizes providers to not deviate from the expected behavior.

The remaining two variables carry user-provided information that relates to the tasks at hand and the progress of their execution. Details are as follows:

- The Task contract carries also information about the *data* required to be stored and/or the *code* to be executed. The way data and code are stored in the contract highly depends on the mechanism of transfer and execution. For this reason, more information on this will be provided in the next sections, where the technicalities of storage and computational tasks are described in detail.
- The *state* variable describes the life cycle of a Task contract. All possible state transitions are depicted in Figure 2. The dashed transitions represent actions performed by the service provider, while the solid transitions represent actions performed by the client.

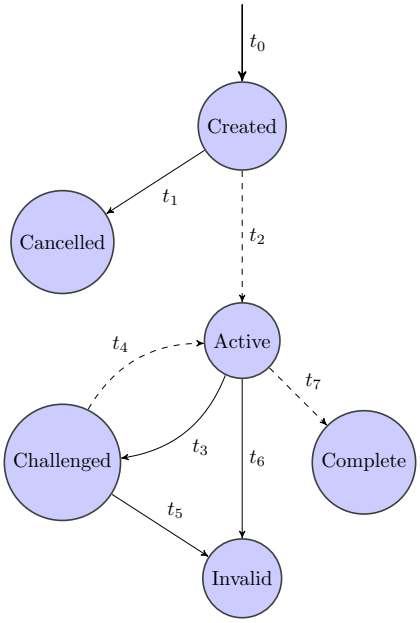

Figure 2. Life cycle of a Task contract.

Transitions t_2, t_4, and t_7 are triggered by the provider. In t_2, the provider receives the data/code and proves that s/he owns it in order to activate the contract. In t_4, the provider proves on-chain that s/he still has possession of the data (or has executed correctly part of the code); in t_7, the provider performs the same proof in order to complete the contract and receive the payment (Step 7 in Figure 1).

Transitions t_0, t_1, t_3, t_5, and t_6, respectively, are triggered by the client. In t_0, the contract is created through the described mechanism. 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 s/he still owns the data. In case the provider fails to give a valid proof, the t_5 transition is triggered by the client and the contract is invalidated; in that case, the collateral is transferred to the client. Finally, the t_6 transition describes the case where the contract is invalidated due to deadline expiration.

4. The Storage Layer

In this section, we describe our approach to implementing decentralized storage over blockchain in CloudAgora. First, we highlight a number of desired properties we want a remote data storage system to have. We then go into detail on how we guarantee those properties in a trust-less environment. Finally, we outline the storage workflow in CloudAgora.

4.1. Considerations Specific to Storage

Although we find a multitude of client requirements regarding remote data storage systems, all these requirements relate to a small number of desired properties we consider essential for such systems. Specifically, in the context of remote storage systems, we mostly consider *Data Integrity*, *Data Availability*, and *Data Confidentiality*. That is the ability to keep our data intact, to access our data at any point in time, and to prohibit unauthorized access.

Switching from a cloud storage model to a decentralized storage model over blockchain requires us to reconsider our approach regarding the above properties. Currently cloud storage systems leverage trust, reputation, and Service Level Agreements (SLAs) in order to guarantee those properties. These mechanisms, however, cannot be used in a trust-less environment.

At this point, it is important to note that the blockchain itself can guarantee those properties. However, as we mentioned before, it is not suitable as a decentralized storage layer. The computational cost of storing large data volumes on-chain is prohibitive. Additionally, since the data have to be replicated in all peers, this approach becomes impractical in a real world scenario. As a result, we

propose an off-chain approach that can guarantee *data integrity, availability, and confidentiality*, using the blockchain only as a distributed ledger resistant to tampering.

4.2. Our Approach

We want CloudAgora to operate in a trust-less environment using a permission-less blockchain. As a result, we observe that it is impossible for the system to enforce any specific user behavior. That is because the participants are free to join or leave the process at any time, or even continue with a different identity. Consequently, we restrict our hypothesis and assume that the participants are rational players, and this allows us to use incentives to guide user behavior and guarantee the desired properties of a remote data storage system mentioned. In practice, we use a combination of incentives and cryptographic tools to ensure those properties.

We identify two distinct roles in a remote data storage scenario, those of a *client* and a *provider*. The *client* is the entity that wants to store his/her personal data to a remote location for safe-keeping, while the *provider* can offer data storage for profit. Note that an entity can have both the role of a *provider* and a *client* simultaneously. Given that through the *market layer* we can match a *client* with a *provider*, in CloudAgora, a storage scenario considers two participants: a *client* and a *provider*. The two roles can interact on- and off-chain, and their on-chain interactions are dictated by a smart contract, which we call the *storage contract*. This contract contains the identities of the *client* and the *provider*, information on the data to be stored remotely, the duration of the contract, and the payment of the *provider*, as decided by the *market layer*. At the time the *storage contract* is activated, the payment amount is transferred from the *client* to the contract and, from that point on, managed by the contract. Based on this configuration, in the following sections, we describe in further detail how we guarantee *Data Integrity, Availability*, and *Confidentiality* on behalf of the *client* in CloudAgora.

4.2.1. Data Integrity

To ensure that the data of the *client* will not be tampered with while at rest, we use two different mechanisms: incentives and Merkle tree proofs.

Assuming the *provider* makes rational decisions, we use incentives to ensure that s/he guarantees the data's integrity. We do so by requesting from the *provider* to transfer to the *storage contract* a collateral amount for the case of data tampering. Consequently, in order for the *provider* to accept the contract s/he has to first transfer to it a previously agreed upon amount. As a result, in case the *provider* fails to prove to the *storage contract* the data's integrity, s/he loses the collateral.

The mechanism that the *provider* uses to prove that s/he owns the *client's* data is based on Merkle trees [19]. The *client*, prior to creating the contract, calculates the Merkle tree of the data s/he wants to store remotely and adds its root hash to the *storage contract*. Similarly, the *provider*, after receiving the *client's* data, verifies his/her integrity by calculating the same root hash and comparing it to the one stored in the contract before accepting it.

At any point in time, the *provider* can send a Merkle proof [19], i.e., a segment of the original file and a list of hashes from the file's Merkle tree, to the *storage contract*, proving that the segment originates from the *client's* data. Since the proof is submitted to the contract, anyone can verify its validity or invalidity by checking if the provided hashes construct the root hash of the data's Merkle tree. Each storage proof uses a randomly selected segment, ensuring 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: A provider storing only x% of the file will be unable to complete approximately 100-x% of the proofs, so the successful periodic demonstration of random segments' possession proves that the provider is very likely storing the whole file. Thus, in CloudAgora we opt for a periodic procedure that involves multiple Merkle proofs of random data segments.

4.2.2. Data Availability

To tackle availability, we ask 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 segment of the data. The *client* can perform the challenge either on- or off-chain.

If the check is performed on-chain and the *provider* fails to reply within the specified time frame, s/he loses the collateral. That way, through the use of monetary incentives, we ensure that the *provider* will 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.

4.2.3. Data Confidentiality

We handle data confidentiality at the *client* side by encrypting the data prior transmission. This approach allows the *client* to choose between different encryption algorithms of even not use any. As with erasure encoding, this process is transparent to the *provider* and the *storage contract*. As a result, encrypted and non-encrypted data are handled identically from all parties.

4.3. Storage Workflow

In this section, we outline in detail CloudAgora's storage workflow. At this point, the *client* through the *market layer's* mechanisms has created and eventually completed a storage auction. Following the storage auction finalization, a *storage contract* is created between the given *client* and a chosen *provider*. In the contract, we store the addresses of the *client* and the *provider*, as well as the root hash of the Merkle tree of the *client's* data, the contract duration, the payment, and collateral amounts. Those amounts are transferred to the contract upon its creation and are 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. 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.

Following the contract creation, the *client* has to upload the data to the *provider*. The *provider* uses a registry contract to register an endpoint to which clients can post data for storage. After receiving a file, the *provider* computes its Merkle tree and verifies its 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, calculated as the contract activation date plus the contract duration, the *provider* can collect the payment. To do so, s/he has to prove that s/he still has the data in her possession. She 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* fails to prove that s/he possesses the *client's* data, the contract transfers all the funds to the *client's* address since it assumes that the data's integrity has been compromised by the *provider*; 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 through the CloudAgora UI. Although it is not possible for the *client* to request the data on-chain, s/he 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 the *provider* gas.

Additionally, while the contract is active, the *client* can challenge the *provider* by requesting a Merkle proof for a specific data segment. 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*, s/he would have to decide beforehand on a number of challenges and store that number

of data segments 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 segment that matches that proof is also verified. Thus, when the *client* challenges the *provider* to prove that s/he owns a specific data segment D, the *provider* sends back to the client the given data segment 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.

When performing an on-chain challenge, the *client* requires from the *provider* to prove that s/he owns a number of different data blocks. Consequently, the *provider* has to send to the *storage contract* a given number of randomly selected data segments as well as the corresponding Merkle tree proofs. The *storage contract* produces the hash of each data segment and combines it with the corresponding proof verifying if the segment 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.

5. The Compute Layer

In this section, we describe how a client can outsource computational tasks to providers without compromising correctness. Providing secure computations by untrusted parties presents its own challenges. For the following discussion, we make two observations:

- In CloudAgora, there is 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 follow an approach that minimizes the on-chain computation load. 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 [20].
- 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 s/he expects to have a monetary profit by doing so.

Based on these observations, we develop a Truebit-like [21] game, where outsourced algorithms are executed off-chain and blockchain is only used for verifying correctness proofs. In such a game, the client has first to announce a task and place a task description on-chain. The task description is part of the Task contract (illustrated in Figure 1) and consists of a JSON file that contains the path where the code is located, the paths for the inputs and any available additional parameter. The declared paths point either to IPFS endpoints or to data stored on chain.

After the task is announced, a solver is selected for executing the task. In addition, any other member of the system may act as a verifier. In the traditional Truebit game, the solver is selected at random. For CloudAgora, we have modified Truebit, and the solver (service provider) is selected based on the auction described in Section 3. The solver performs computations off-chain, and only after completion does s/he reveal the solution on the blockchain. Verifiers compute the solution in private and check if the result they produce agrees with the solution that the solver proposes. In the case of agreement, the solver is paid and the game stops. Otherwise, the verifier can challenge the solver, and an interactive proof takes place on-chain. If the solver proves to be malicious, the verifier receives the solver's reward and the solver loses the deposited collateral we discussed in Section 3. If the verifier has triggered a false alarm, s/he is obliged to pay for the resources wasted due to the interactive game.

Interactive proof. In Truebit, disputes between solvers and verifiers are resolved through an interactive game. Although we keep this mechanism intact, in favor of completeness, we provide a short description of the way these proofs work. First of all, we must ensure 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. 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, and machine state). The game also determines a parameter c that indicates how many configurations the solver broadcasts to the blockchain in each round 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. S/he 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 the 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. A more detailed analysis of this process can be found in [21].

5.1. Discussion on Applicability

Typically, techniques that provide verifiable computations are restricted to work only over a limited set of applications [21–23]. In contrast to this, TrueBit's protocol can process arbitrarily complex tasks. By allowing heavy computations to run off-chain and use the blockchain only for providing proofs, Truebit offers a scalable, state-of-the-art solution to verifiable computing.

In a Truebit-based platform such as CloudAgora, users may outsource any application written in C, C++, or Rust. The Truebit infrastructure then provides compilers that turn the high-level code of the user into a web-assembly format (WASM). This happens in order to allow the miners to run parts of the code in case of a dispute.

Thus, as CloudAgora employs Truebit for executing computational tasks, it is only as general as Truebit is; i.e., it is restricted to centralized single-threaded applications. As we next explain, in our future plans, we intend to extend this model to also work in distributed data parallel frameworks, such as Apache Spark.

6. Extensions to Support Distribution

The description of CloudAgora has so far focused mainly on the use case scenario where the client needs to perform a centralized storage or computational task, i.e., a task that requires a single provider in order to be executed. Clients, however, may explicitly opt for a distributed infrastructure both for storage and for processing for availability, fault tolerance, and performance reasons.

CloudAgora is thus extended to allow for this scenario, where a distributed environment is required. As far as distributed storage goes, the extensions support the distribution of a data item to multiple providers either as a whole or in chunks. Such a practice ensures data availability in the presence of unreliable nodes and recovery in the face of unstable, high churn environments through redundancy.

For distributed computation, CloudAgora is extended to support data parallel tasks, i.e., operations performed in different nodes over non-overlapping parts of the input data. Supporting distributed frameworks for storage (e.g., HDFS) and computation (e.g., Hadoop MapReduce, and Spark) that will be able to run on top of ad hoc clouds formed through CloudAgora is a subject of future work.

In the following, we describe in detail the necessary extensions to all layers of the CloudAgora architecture.

6.1. Extensions to the Market Layer

As described in Section 2, the market layer focuses on the description of the client's Task, be it a storage or a computation one, and the selection of the most suitable provider to perform it through an open auction process. Interested parties place their bids and the lowest one wins.

When the task is a distributed one, multiple providers must be involved. Initially, the client must specify the number of providers required upon a new distributed task submission. S/he then partitions the data into the corresponding number of segments and calls the AuctionFactory contract. AuctionFactory needs to be enriched with a new variable *numOfProviders* in order to generate an appropriately parameterized Auction. Moreover, the *task difficulty* parameter of the AuctionFactory contract now reflects the corresponding difficulty of the task part that each provider will undertake. As we assume that the task is equally divided among participating providers, we expect difficulty to be proportional to the size of each file segment.

During the bidding phase, the *numOfProviders* value determines the number of lowest bids that need to be maintained at all times: For k number of providers required, the Auction contract needs to keep the k lowest bid values and the corresponding provider addresses. After the auction is finalized, the Auction contract automatically constructs k Task contracts, and each of them binds a specific provider with a specific block of the initial file. The described extensions are illustrated in Figure 3.

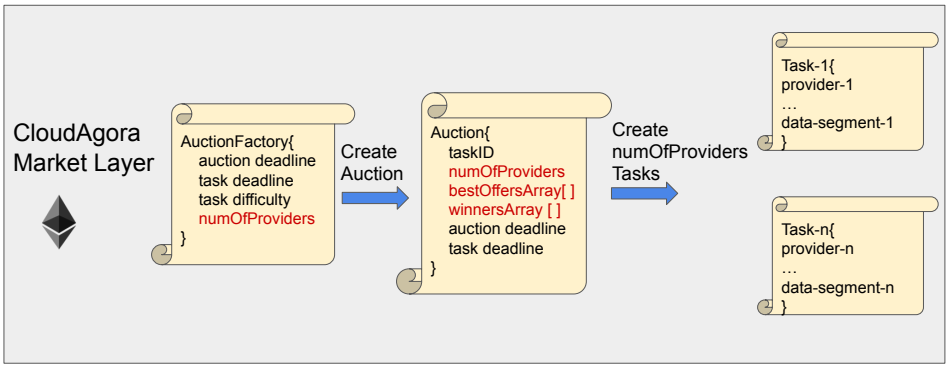

Figure 3. Extensions in the market layer to support distributed jobs.

6.2. Extensions to the Storage Layer

One of the main reasons for the choice of multiple storage providers is fault tolerance: A client must be able to access and possibly recover her data even in the event of failures.

The incentives provided for data availability and integrity, as described in Section 4, can be considered sufficient to cover the case of data loss. In such a case, the *provider* will not be able to prove that s/he owns the original data and eventually lose his/her collateral. However, given that the Merkle proof mechanism can only guarantee the availability of a percentage of the original data, CloudAgora is extended to 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 m of the n fragments, where $m < n$. There is a large collection of erasure codes available today. We opted for the Reed-Solomon codes [24] because of their popularity and wide use.

Following the above process, the client data are erasure-encoded, i.e., split into n encrypted fragments and stored across n providers. This allows a client to attain high data availability even if providers are not generally reliable since only m out of n pieces are needed to recover the data, thus only m out of n providers need to be reachable at any point in time. Moreover, by erasure encoding the

data before remotely storing them, the *client* ensures that retrieving only part of them is sufficient to restore them in their entirety.

$$P(D \leq n - m) = e^{-\lambda} \sum_{i=0}^{n-m} \frac{\lambda^i}{i!}$$

Let us define the *expansion factor* of an erasure code as the ratio $\frac{n}{m}$. The analysis then provides the following Table 1:

Table 1. Durability of erasure-codes for $p = 10\%$ and various (n, m) parameters.

m	n	Expansion Factor	$P(D \leq n - m)$
4	6	1.5	97.688471224736705%
4	12	3	99.999514117129605%
20	30	1.5	99.970766304935266%
20	50	2.5	99.999999999999548%
100	150	1.5	99.999999999973570%

Interestingly, by following the above analysis, it follows that the durability of a $(m = 20, n = 40)$ erasure code is better than a $(m = 10, n = 20)$ erasure code, even though the expansion factor $(2\times)$ is the same for both. This is because, in the first case, the risk is spread across more nodes. Thus, choosing the "right" block size to split a data file may have a great impact on storage durability.

Moreover, erasure codes can speed up the downloading process of a data item, since a download can run in parallel from the m providers with the highest upload rate. Thus, the client can reduce latency and maximize available bandwidth.

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.

6.3. Extensions to the Compute Layer

The distribution of a computational task is usually favored when trying to speed up data parallel operations. Instead of having one provider operating over the whole dataset, one can split the dataset into k chunks and have each of the k providers perform the same computation over each of the k chunks in parallel.

After the selection of the k providers upon completion of the auction process, k compute task contracts are deployed, one per provider. These contracts are independent of each other and function as regular one-to-one contracts. The results of each provider's computation is collected by the client. In the current implementation, if merging intermediate results in a MapReduce style is required, it is the responsibility of the client to collect them and run the *reduce* function.

7. Discussion

In this section, we discuss the best practices when choosing some important application parameters, pointing out the trade-offs they entail. The first set of parameters includes the frequency of invoking the Merkle proof verification process in combination with the total number of data blocks that constitute the Merkle tree leaves. The second set of parameters relates to the number of erasure shares as well as the redundancy degree of erasure coding to ensure a certain level of reliability when storing data.

7.1. Data Verification Parameters

As presented in Section 4, a storage challenge can take place either on- or off-chain and is triggered by the client. During such a challenge, the provider needs to present the storage contract or the client (for on- and off-chain verification, respectively) the contents as well as the hashes of sibling nodes along the path from the leaf to the root of the Merkle tree for a specific block of the original data,

as dictated by the client. This entails an a priori choice and local maintenance of the random data blocks with which the client will challenge the provider. The greater the coverage of the original data requested during the challenges, the stronger the guarantee that the data has not been tampered with by the host and can be retrieved at any point in time.

However, the verification process has a non-negligible overhead and, when performed on-chain, costs gas. Thus, when it comes to the number of storage challenges, or equivalently the frequency of the challenge initiation, there is clearly a trade-off between cost and security. One would be tempted to divide the data into a few large blocks, create a Merkle tree with a limited number of leaves, and initiate a small number of challenges that cover a large portion of the original file. However, in the case of off-chain verification, this would mean that the client would have to maintain the challenged blocks (and thus a large percentage of the file) locally, which defies the purpose of a remote storage, while on-chain verification would cost more in time and gas due to the large size of the file block that needs to be processed by the contract.

Thus, the best practice is to construct Merkle trees with a large number of leaves, i.e., smaller sized blocks, and randomly challenge a small but adequate percentage of them.

7.2. Erasure Coding Parameters

The erasure coding strategy guarantees access to the data even in the event of node failures. In erasure coding, one can control the *expansion factor*, i.e., the degree of data redundancy by choosing the right n and m parameters, as described in Section 6. More specifically, the expansion factor is calculated as n/m, where m out of n erasure fragments are required to restore a file. In the special case of $m = 1$, erasure coding is degraded to simple replication, with a replica factor of n.

Obviously, a greater expansion factor of an erasure code scheme increases the availability of the file. Unfortunately, it also increases the storage overhead and associated cost. However, the expansion factor alone is not the ultimate indication of the availability of the stored file. The absolute number of erasure fragments plays a decisive role. Among schemes with the same expansion factor, the larger the number of file fragments scattered among the provider nodes, the higher the probability that m of them will be reachable at any time.

Thus, to avoid excessive costs due to high redundancy, the best practice is to opt for a low expansion rate with adequately high values of n and m, which will achieve the desired availability.

7.3. The Case for a Permissioned Blockchain

Considering our motivation, i.e., breaking the cloud computing market monopoly through the use of blockchain technology, we have opted, in our system design, for a permission-less blockchain. This decision contributed to a strongly decentralized design, since anyone can become a miner leaving our system vulnerable only to 51% attacks. However, since the system we propose remains blockchain-agnostic there are a number of trade-offs to be discussed in the case of implementing CloudAgora on top of a permissioned blockchain.

- A permissioned blockchain implementation will allow for considerably faster transaction times; as a result, storage and compute tasks will be launched faster. However, we argue that the duration of a compute or storage task is dominated by the task itself and not the time spent for on-chain transactions.
- Since a permissioned blockchain does not require a proof-of-work type of consensus algorithm, it can have a much lower energy footprint. Although this is something to consider, there is abundant ongoing research on consensus algorithms for public blockchains based on a proof of stake, proof of capacity, proof of elapsed time, etc., which aim at improving upon the energy efficiency of permission-less blockchains [25]. In fact, Ethereum is in the process of switching to a proof-of-stake system [26].

- Finally, we argue that, by building CloudAgora on top of a permissioned blockchain, we defy our initial purpose of democratizing cloud computing. A monopoly will still be formed by the entities that control the blockchain and, as a result, will be able to drive the cloud computing market.

8. Prototype Implementation

We implemented a prototype of CloudAgora as a DApp over Ethereum. As a result, the contracts that govern the auction and the on-chain *client*, *provider* interactions are in Solidity [27]. All the off-chain logic is implemented in NodeJS using web3.js to interface with the Ethereum blockchain. The CloudAgora GUI is implemented as a WebApp using koa [28] and Angular [29]. We used parts of the Truebit codebase to implement the compute module of CloudAgora, while the Storage is implemented from scratch. The code is freely available as an open source project (GloudAgora github link: https://github.com/tabac/cloudagora). Figure 4 shows a snapshot of the user interface of our DApp. In the dashboard, the client (provider) can inspect the submitted (received) tasks along with the corresponding payment amounts and deadlines.

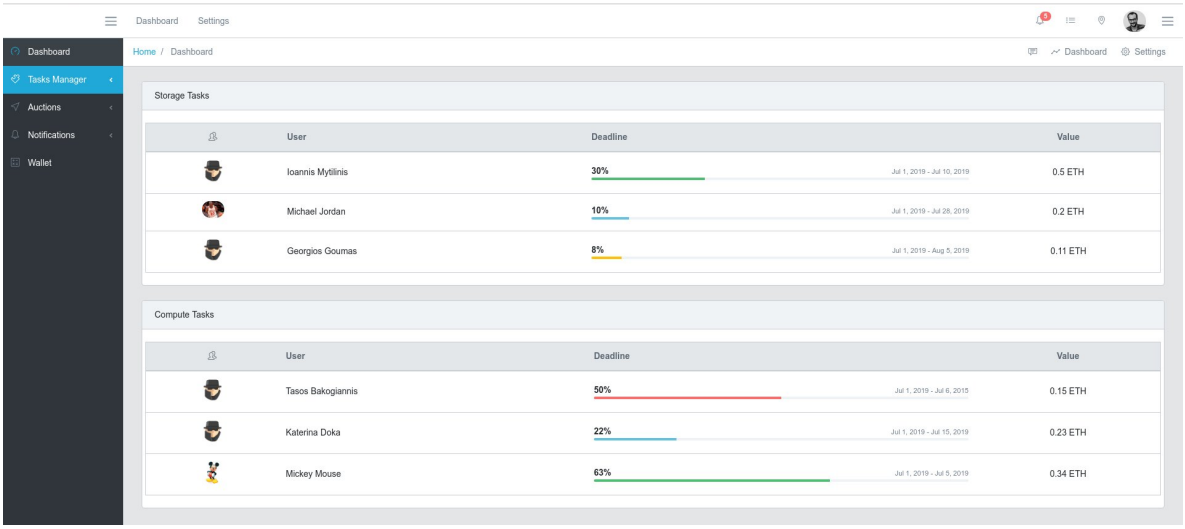

Figure 4. Dashboard of the CloudAgora DApp.

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 committing 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 [30].

9. Related Work

The proposed platform enables users to enjoy data hosting and task execution services at a low cost, by breaking the monopoly of large cloud computing providers and allowing anyone to provide idle resources. In such an untrusted environment, compliance to the policy agreed upon by both the consumer and the provider of the service is safeguarded through the use of smart contracts, which publicly verify the correctness of the outsourced tasks and automatically trigger payments. This model of *decentralized clouds* opens new opportunities and has recently gained attention by both academia and industry. In [31], there is a survey that discusses the pros and cons of both traditional cloud deployments and blockchains and comes up with a proposal for the architecture that decentralized clouds should have. CloudAgora's design is in accordance with the main principles presented in [31]. Similar approaches in the competition landscape include blockchain-based projects that offer storage and/or compute services.

In the domain of storage provisioning, 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 [16] supports smart contracts between hosts and renters, which are stored and executed in a custom blockchain. Merkle proofs are used to easily check the integrity and existence of a piece of data on a remote host. Storj [14] is based on the Kademlia Distributed Hash Table (DHT) [32] 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 [15] relies on zk-SNARK [33], a cryptographic tool for zero-knowledge verifiable computation, to provide the so-called *Proof-of-SpaceTime*, i.e., a proof that a piece of 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 data hosting, while CloudAgora provides both data hosting and computational services.

The combination of storage and processing is covered by projects such as GridCoin, Enigma, Golem, Dfinity, and iExec. The GridCoin [34] 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. Due to its pure focus on volunteer grid computing projects, it is mainly limited to altruistic sharing of resources for scientific research.

The Enigma [18] platform adopts multi-party computation (MPC) [35] protocols to guarantee both the correctness of execution and data privacy preservation. To allow nodes to operate on encrypted shards of data, it utilizes homomorphic encryption, thus limiting its support to only specific types of computation. This fact significantly narrows down the use cases where Enigma can actually be used, hindering its wide adoption in practice.

Golem [17] relies on 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. Contrarily, CloudAgora aims to support any type of task. Moreover, while CloudAgora utilizes Truebit as a means of verifying the correctness of the outsourced computation as well, the reputation mechanism and it does not employ it as is, but rather changes the random way in which tasks are allocated to solvers by adding an auction game that matches resource requests to providers according to cost criteria.

Dfinity [36] is essentially a new blockchain that, as the creators claim, aims to realize the "Internet computer," a distributed processing substrate that will replace smart contract supporting blockchains and power the next generation of distributed applications. iExec [37] 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 already a popular and widely adopted blockchain.

10. Conclusions

The public cloud market has become a monopoly, where a handful of providers—which are by default considered trusted entities—define the prices and accumulate knowledge from users' data and computations. To tackle these drawbacks, in this paper, we present CloudAgora, a blockchain-based system that enables the provision of remote storage and computing infrastructures in an ad hoc and affordable manner. Users of CloudAgora act either as providers, offering idle resources, or as clients, requesting resources, both centralized and distributed. Each resource request is implemented as a smart contract that sets the rules of an open auction, accepting bids by any potential provider. The provider, or set of providers in the case of a distributed task, is chosen based on the height of the placed bids. The agreement between providers and consumers creates a new smart contract, which allows for traceability of actions and automatic triggering of payments. While storage and processing per se

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 both off- and on-chain. Our prototype implementation relies on Ethereum, the most prevalent blockchain supporting smart contracts, although any blockchain with such a functionality can be used.

Author Contributions: Investigation, Software and Writing—original draft, T.B., I.M., K.D.; Writing—review and editing, I.M., K.D. and G.G.; Conceptualization, K.D.; Validation, K.D.; Supervision, G.G.; Methodology, T.B., I.M., K.D. and G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was 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).

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

References

1. How cloud computing is changing the world ... without you knowing. Available online: https://www.theguardian.com/media-network/media-network-blog/2013/sep/24/cloud-computing-changing-world-healthcare (accessed on 7 February 2020).
2. Cloud computing: a business model game changer. Available online: https://www.theguardian.com/media-network/media-network-blog/2012/mar/30/cloud-computing-business-model (accessed on 7 February 2020).
3. Boss, G.; Malladi, P.; Quan, D.; Legregni, L.; Hall, H. Cloud computing. *IBM white paper* **2007**, *321*, 224–231.
4. 451 Research. Available online: https://451research.com/images/Marketing/press_releases/Pre_Re-Invent_2018_press_release_final_11_22.pdf (accessed on 7 February 2020).
5. Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper. Available online: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html (accessed on 7 February 2020).
6. Report Linker, Cloud Computing Market by Service, Deployment Model, Organization Size, Workload, Vertical And Region - Global Forecast to 2023. Available online: https://www.reportlinker.com/p05749258/Cloud-Computing-Market-by-Service-Deployment-Model-Organization-Size-Workload-Vertical-And-Region-Global-Forecast-to.html. (accessed on 7 February 2020).
7. Canalys: Cloud infrastructure spend grows 46% in Q4 2018 to exceed US$80 billion for full year. Available online: https://www.canalys.com/static/press_release/2019/pr20190204.pdf. (accessed on 7 February 2020).
8. Wang, H.; Jing, Q.; Chen, R.;He, B.; Qian, Z.; Zhou, L. Distributed systems meet economics: pricing in the cloud. Available online: https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/WangH.pdf (accessed on 7 February 2020).
9. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system **2008**. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 7 February 2020).
10. Swan, M. *Blockchain: Blueprint for a new economy*; O'Reilly Media, Inc.: Sebastopol, USA, 2015.
11. Croman, K.; others. On scaling decentralized blockchains. In proceedings of the International Conference on Financial Cryptography and Data Security, Heidelberg, DE, 31 August 2016; pp. 106–125.
12. Eberhardt, J.; Tai, S. On or off the blockchain? Insights on off-chaining computation and data. In proceedings of the European Conference on Service-Oriented and Cloud Computing, Cham, CH, 1 September 2017; pp. 3–15.
13. Doka, K.; Bakogiannis, T.; Mytilinis, I.; Goumas, G. CloudAgora: Democratizing the Cloud. In proceedings of the International Conference on Blockchain, Cham, CH, 19 June 2019; pp. 142–156.
14. Wilkinson, S.; Boshevski, T.; Brandoff, J.; Buterin, V. Storj a peer-to-peer cloud storage network. Available online: https://storj.io/storj2014.pdf (accessed on 7 February Year).
15. A robust foundation for humanity's information. Available online: https://filecoin.io/. (accessed on 7 February Year).
16. Vorick, D.; Champine, L. Sia: Simple decentralized storage. Available online: https://sia.tech/sia.pdf. (accessed on 7 February Year).
17. Available online: https://golem.network/ (accessed on 7 February Year).

18. Zyskind, G.; Nathan, O.; Pentland, A. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint* **2015** *arXiv:1506.03471*.
19. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In proceedings of the Conference on the Theory and Applications of Cryptographic Techniques, Heidelberg, DE, 16–20 August 1987; pp. 369–378.
20. 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, CO, USA, 12–16 October 2015; pp. 706–719.
21. Teutsch, J.; Reitwießner, C. A scalable verification solution for blockchains. Available online: https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf (accessed on 7 February 2020).
22. Ben-Sasson, E.; Chiesa, A.; Genkin, D.; Tromer, E.; Virza, M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In proceedings of Annual Cryptology Conference, CA, USA, 18–22 August 2013; pp. 90–108.
23. Parno, B.; Howell, J.; Gentry, C.; Raykova, M. Pinocchio: Nearly practical verifiable computation. *2013 IEEE Symp. Secur. Priv.* **2013**, 238–252. doi: 10.1109/SP.2013.47
24. Reed, I.S.; Solomon, G. Polynomial Codes Over Certain Finite Fields. *J. Soc. Ind. Appl. Math.* **1960**, *8*, 300–304.
25. Bano, S.; Sonnino, A.; Al-Bassam, M.; Azouvi, S.; McCorry, P.; Meiklejohn, S.; Danezis, G. SoK: Consensus in the age of blockchains. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies, NY, USA, 21–23 October 2019; pp. 183–198.
26. Ethereum 2.0 Specifications. Available online: https://github.com/ethereum/eth2.0-specs (accessed on 7 February 2020).
27. Solidity, the Contract-Oriented Programming Language. Available online: https://github.com/ethereum/solidity (accessed on 7 February 2020).
28. koa, next generation web framework for node.js. Available online: https://koajs.com/ (accessed on 7 February 2020).
29. Angular, One framework, Mobile & Desktop. Available online: https://angular.io/ (accessed on 7 February 2020).
30. 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, NY, USA, 14–19 May 2017; pp. 1085–1100.
31. Westerlund, M.; Kratzke, N. Towards Distributed Clouds: A Review About the Evolution of Centralized Cloud Computing, Distributed Ledger Technologies, and A Foresight on Unifying Opportunities and Security Implications. *2018 Int. Conf. High Perform. Comput. Simul. (Hpcs)*, **2018**, 655–663. doi: 10.1109/HPCS.2018.00108
32. Maymounkov, P.; Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In proceedings of the International Workshop on Peer-to-Peer Systems, Heidelberg, DE, 10 October 2002; pp. 53–65.
33. Ben-Sasson, E.; Chiesa, A.; Tromer, E.; Virza, M. Succinct non-interactive zero knowledge for a von Neumann architecture. Available online: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson (accessed on 7 February 2020).
34. GridCoin White Paper. Available online: https://www.gridcoin.us/assets/img/whitepaper.pdf (accessed on 7 February 2020).
35. Goldreich, O. Secure multi-party computation. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, NY, USA, 22–24 May July 1996.
36. dfinity: The Internet Computer. Available online: https://dfinity.org/ (accessed on 7 February 2020).
37. iExec Blockchain-Based Decentralized Cloud Computing. Available online: https://iex.ec/ (accessed on 7 February 2020).