Trustless Collaborative Cloud Federation

Bishakh Chandra Ghosh* Sandip Chakraborty†

November 15, 2024

Abstract

Multi-cloud environments such as OnApp and Cloudflare have turned the cloud marketplace towards a new horizon where end-users can host applications transparently over different cloud service providers (CSPs) simultaneously by taking the best from each. Existing cloud federations are typically driven by a broker service which provides a trusted interface allowing the participant CSPs and end-users to coordinate. However, such a broker has the limitations of any centralized trusted authority like risk of manipulation, bias, censorship, single point of failure, etc. In this paper, we propose a decentralized trustless cloud federation architecture called *CollabCloud* which eliminates any central mediator while addressing the challenges introduced by byzantine participants. *CollabCloud* utilizes blockchain, and introduces a novel interoperability protocol bridging a permissionless blockchain as an open interface for the end-users, and a permissioned blockchain as a coordination platform for the CSPs. We have implemented *CollabCloud* with Ethereum, Hyperledger Fabric and Burrow platforms. Experiments with a proof-of-concept testbed emulating 3 CSPs show that *CollabCloud* can operate within an acceptable response latency for resource allocation, while scaling upto 64 parallel requests per second. Scalability analysis over Mininet emulation platform indicates that the platform can scale well with minimal impact on the response latency as the number of participating CSPs increases.

1 Introduction

With OnApp Federation (`https://onapp.com/onapp-federation/`)[1] being one of the largest private-public cloud marketplace having more than 55 compute and 170 CDN locations spanning across 113 cities of 43 countries (as of March 24, 2022), federation based multi-cloud environments have recently seen a growing popularity. In a federated multi-cloud architecture, multiple public and private

*Bishakh Chandra Ghosh is with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721302, India (e-mail: ghoshbishakh@gmail.com).

†Sandip Chakraborty is with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721302, India (e-mail: sandipc@cse.iitkgp.ac.in).

[1]All URLs are last accessed as on March 26, 2022

cloud service providers (CSPs) get connected over a single marketplace, where they can buy and sell cloud resources on-demand. Such architecture helps the CSPs to increase the overall revenue and performance by selling their spare capacity, scaling up the capacity by procuring resources from others, and enabling fluidity by allowing the workloads to get migrated across clouds depending on the resource demands [1–3]. Traditionally, all the interactions among different CSPs and the end-users take place through a centralized cloud broker or a marketplace (Figure 1(a)) which is considered to be a trusted authority [1, 4] and is responsible for ensuring compliance of the CSPs to the *federation level agreement* (FLA) – a set of policies that regulate the sharing of resources and data among the CSPs and ensure the sustainability of the federation through fair profit sharing. In an *infrastructure-as-a-service* (IaaS) cloud marketplace, the end-users request for the cloud infrastructure in the form of virtual machine (VM) requests from the federation, and the broker coordinates among the CSPs to allocate the VM instances to the end-users.

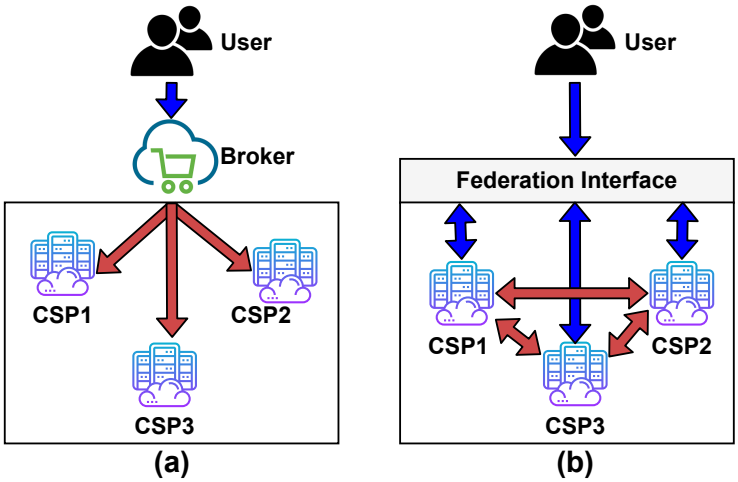


Figure 1: (a) Broker-based and (b) *CollabCloud* Federation

However, such centralization through a cloud broker or a common marketplace brings up the following issues.

1. **Mutual trust among the broker and the CSPs:** The CSPs need to trust the broker for maintaining the FLA. Any discrepancy between the buyer and the seller needs to be resolved by the broker.
2. **Bias towards established federations:** Because of the above trust issue, any new CSP would always prefer to join in an already established federation – this results in a market bias towards the large and established brokers.
3. **Pricing control:** In a central architecture, the resource prices are typically being controlled by the broker. Moreover, the broker being a business entity itself also tries to maximize its own revenue. For example, the

OnApp Federation fixes the price of the resources under three tiers – *Dev Tier*, *Pro Tier* and *Enterprise Tier*[2]. During subscription, the sellers need to publish the resource pricing solely based on the tier requirements[3].

4. **Centralized interface:** The broker acts as a centralized interface for end-users to access the entire federation. Workloads from end-users are scheduled among individual CSPs by the broker which in turn may impact the revenue of individual CSPs.
5. **Single point of failure:** The entire federation goes down if there is a failure at the broker. The CSPs need to depend on the services provided by the broker.

The above mentioned limitations of a broker-based cloud federation can be tackled through a decentralized federation by removing the broker from the architecture. However, in a cloud marketplace, the CSPs compete with each other to maximize individual profits; therefore, the system needs to be run in a trustless setting. In order to design a fully trustless decentralized architecture for cloud federations, protocols must be developed to achieve all the functionalities provided by the broker which otherwise acted as the trusted central point of coordination. Decentralizing each of the broker functionalities of a typical cloud broker has its own set of challenges.

(a) Unified interfacing in a trustless byzantine setting: In the absence of a cloud broker, a unified federation interface is required to enable the end-users to view the catalog, query prices, request for resources, get the resource access information and credentials, and make payments. However, the users as well as the CSPs can behave maliciously; therefore, the information provided by the unified federation interface should be verifiable by all the stakeholders of the federation.

(b) Agreement on federation: Secondly, the cloud broker in a federation maintains agreement among all the CSPs in terms of federation structure (participants, their roles and policies), resource catalog and resource scheduling. In case of a brokerless setup, all the CSPs must reach consensus on all the interactions among the end-users and the CSPs in the federation. Simply broadcasting the user requests to the CSPs will not work, since all CSPs must reliably receive the same set of VM requests from the end-users in the same order to process them correctly. Moreover, the end-users or the CSPs may try to deceive the federation by exhibiting byzantine behavior.

(c) Fair scheduling of user requests: Thirdly, once an end-user requests for a cloud VM from the federation, the broker in a centralized marketplace decides which CSP is going to serve that particular request, based on a predecided protocol or policy. The CSPs trust the broker to execute the protocol honestly and fairly. Thus, in the absence of a broker, protocols need to be designed such that CSPs come to a consensus on the scheduling of the end-user requests. The scheduling policies must be developed such that the federation provides fair opportunities and incentives to all participating CSPs.

[2] `https://docs.onapp.com/federation/market-pricing`

[3] `https://docs.onapp.com/federation/sellers/seller-pricing-tiers-for-compute-zones`

(d) Securing the access to the resources in an open environment: Finally, once a resource request from an end-user is scheduled and the federation has decided the resource allocation, the credentials of the allocated VM instances need to be passed to the end-user while maintaining transparency.

Towards such a brokerless decentralized cloud federation, we develop a distributed ledger technology (DLT)-based architecture ***CollabCloud***, which leverages a combination of pubic and private blockchain technologies [5, 6] to enable a trustless decentralized cloud federation. In an initial version of the paper published in IEEE INFOCOM 2021 [6] we developed a generalized framework for public-private blockchain interoperability to transfer assets and data from a public (permissionless) blockchain to a private (permissioned) blockchain and vice-versa. We presented an initial prototype of a decentralized cloud federation in that paper by leveraging the above interoperability architecture. This paper extends the initial prototype to a fully developed collaborative brokerless multi-cloud federation platform called *CollabCloud* by addressing the challenges mentioned above in detail. We provide the execution workflow of different smart contracts to facilitate a multi-cloud federation platform. In addition, we discuss the correctness properties of *CollabCloud*, and thorough performance comparison and benchmarking under different workloads.

In *CollabCloud*, the end-users of the federation interface with a public blockchain network to submit the VM requests which are then passed to the CSPs to collectively and transparently decide the scheduling of VM requests among themselves. The CSPs in *CollabCloud* coordinate over a private blockchain network to decide about the request scheduling as well as to maintain the federation-level management activities. However, one of the major design challenges in this architecture comes from the fact that the two different blockchain networks run their own consensus primitives independently [7], and therefore, there is a requirement to develop an interfacing between the two. *CollabCloud* designs a set of smart contracts over the two blockchains through which the consensus on one network is transferred to the other. Further, *CollabCloud* designs a secure way to transfer the VM credentials from the CSPs to the end-users using public key authentication of `SSH Protocol` [8] and leveraging the collective signing (CoSi) technology [9]. We have done a proof-of-concept implementation of *CollabCloud* using Ethereum as the public blockchain platform and Hyperledger Fabric and Burrow as the two different candidates for the private blockchain platform and tested it with three emulated CSPs. The experiments indicate that the decentralized functionalities of *CollabCloud* do not incur much overhead on the federation, and a Mininet-based emulation with 32 CSPs also ensures its scalability over a large geo-distributed setup.

2 Related Work

Cloud federations are meant to provide *"one stop shopping"* for transparently utilizing multiple offerings from multiple CSPs [10]. Towards this goal, the primary hurdle that has been faced and then addressed to a large extent is the

interoperability between different CSPs through some common open standards or protocols [11–13]. Although not completely centralized, these architectures where the CSPs are independent and independently coordinate with other CSPs are less like 'federations', and more like 'interclouds' [14, 15]. The interaction between any two participants in an intercloud architecture remains private between the pair; consequently, there is no federation level policies or agreements applied as a whole. Moreover, the end-users see the CSPs as separate providers and there is also no unified scheduling policy among the federation participants.

In a federation, the service requests from the end-users or cloud customers should be scheduled according to some pre-agreed protocol, and there must be a single interface for the customers to communicate with. In order to achieve these functionalities, several cloud-broker based architectures, similar to Figure 1(a), have been introduced [4, 16–19]. In these works, the broker is given the responsibility to act as the glue between different participant CSPs, as well as between the end-users and the federation.

Blockchains have been used to provide security, privacy and transparency to various aspects of clouds and cloud federations, like access control and auditing for cloud storage [20, 21] and cloud federations [22], multi-user cooperative search [23], data provenance [24], budget allocation [25], etc. An essential building block required for any decentralized federation is storage. CloudFS [26] secures cloud storage services using blockchain, whereas ContractChecker [27] ensures storage consistency. Another important aspect is decentralized identity management which can be achieved through decentralized identifiers (DID) and verifiable credentials (VC) [28, 29].

Comprehensive federation architectures have since been introduced like the SUNFISH Federation-as-a-Service architecture [30] which focuses on distributed governance involving federation members. In order to enforce such federation governance over SUNFISH, Margheri *et al.* proposed to use blockchain [31] because of its strong integrity and immutability guarantees. However, they do not provide any specifications of the design of governing smart contracts on blockchain. Moreover, for processing and scheduling end-user requests, SUNFISH relies on Intelligent Workload Manager, a component acting as the service broker, and do not have any decentralized architecture for the same.

Savi *et al.* introduced a public blockchain based brokerage platform [32] using Ethereum through which spare fog resources can be advertised, selected and paid for using cryptocurrencies. Although decentralized, here the blockchain brokerage platform acts as a public marketplace where the providers can list their resources, which results in separate catalogs and no unified federation policies. Furthermore, there is no protocol provided for securely transferring access to the procured resources. [33] introduces a decentralized exchange based cloud federation where all the CSPs act independently, and the federation acts as an underlying layer that enables sharing of IaaS resources through a private blockchain-based marketplace. Although this architectures is brokerless, the end-users of the system do not become aware of the overall federation and only see the CSPs as individual providers like the interclouds. As a result, there is no unified inteface to the federation which is crucial for the user experience.

3 System Model and Design Challenges

The primary design challenge in *CollabCloud* is to eliminate any central point in all possible interactions between the participants of the federation, such as, (a) service discovery, (b) service request, (c) collective processing of the request and (d) service delivery. We consider the interconnecting network to be partially synchronous where there is an upper bound Δ on the time of message delivery [34, 35]; if a message is not received within the time bound Δ, then it is considered as a message fault. The intuition is that in a realistic communication environment, the messages must have arbitrary but bounded delay. Additionally, we consider different types of attacks which might affect the above operations, as follows.

3.1 Attack Model

A decentralized cloud federation is prone to following types of attacks, which we take care of in the design of *CollabCloud*.

Byzantine participants: Participants in the system, both CSPs and cloud users, may exhibit byzantine behavior [36]. An end-user can try to deceive the federation by sending different requests to different CSPs, while the CSPs can themselves collude to alter the results of the scheduling protocol and take control of the federation.

Sybil attacks: BFT consensus protocols mostly rely on one very important assumption that each participant has only one distinct identity which is known to every other participant [37]. If a participant is able to use multiple identities, then using such redundancy it can launch a "Sybil Attack" [38]. The end-users of a cloud can launch such Sybil attacks by using multiple identities.

Impersonation attacks: In a decentralized federation, no single CSP can be responsible to communicate with its end-users. As a result, the users do not have any single trusted source of information that guarantee to provide the decisions of the federation. Exploiting this, a malicious CSP might try to deceive an end-user by posing as the federation and providing false information. Thus there must be some mechanism for the end-users to verify that the information they receive has been collectively generated by the agreement of the federation participants.

Leakage of sensitive information: In the absence of any trusted broker, there is no secure channel between the CSPs and the end-users for communication. Instead, the information has to be disseminated through consensus protocols across the decentralized network. As a result, sensitive information like VM access details might get leaked.

3.2 Design Philosophy

CollabCloud's primary objective is to provide a brokerless cloud federation while maintaining all the functional and non-functional benefits of a cloud federation. In order to do so, it must provide the following key functionalities.

1. **Brokerless Federation Interface**: This should provide (a) an interface for providing unified catalog and information to end-users, (b) an interface through which end-users can request for resources (VMs), and (c) protocols for transferring resource access from the federation to the end-users.
2. **Brokerless Collaboration**: This ensures (a) CSPs collaboratively come to consensus upon catalog, pricing, and service level agreement of the federation, (b) scheduling of end-user requests to different participating CSPs, and (c) protocols to generate federation messages only upon consensus of the participants, which the end-users should be able to verify.

3.2.1 Brokerless Federation Interface

Without the broker, the cloud federation needs an alternative interface to interact with its end-users. This unified interface is essential to represent the federation as a larger entity (like an organization; for example, similar to the OnApp federation marketplace). We can compare it to the large e-commerce platforms, where the sellers are different smaller businesses which are hosted under a single umbrella. The e-commerce platform provides them the unified interface to the end-users (customers), and also handles the scheduling of the requests (which business is chosen for a particular product).

Although at the first glance providing such an interface seems trivial, there are indeed multiple challenges that needs to be taken care of. Simple solutions like an API gateway will lead to centralization, whereas message broadcast from the end-users to the CSPs will not work since the communication environment does not guarantee all the CSPs to receive the same messages in the same order. Furthermore, byzantine behavior of the participants means messages might get colluded or tampered. Therefore, in *CollabCloud*, the following two guarantees need to be ensured at the federation interface.

Definition 1. (Federation Interface Safety) The federation interface should ensure that all the correct CSPs agree on the same set of incoming end-user requests, and also agree on the same ordering of the requests.

Definition 2. (Federation Interface Liveness) The federation interface must ensure that all the correct end-user requests are eventually received by the federation.

Thus, a mechanism is needed such that the the safety and liveness guarantees are met by the interface, and the CSPs are in consensus on each request. This can be mapped to the traditional atomic broadcast problem [39] which is equivalent to the consensus problem. BFT protocols have been extensively studied in the literature and are widely used in private blockchains [36, 37, 40]. However, in such a private blockchain system, all the participants must be known to each other, and each participant can uniquely identify every other participant. This requirement poses a barrier for the end-users of a cloud federation, as it would

be a significant overhead to verify the identity of each end-user in a decentralized setting; otherwise, the adversarial end-users may launch a sybil attack [38] as mentioned earlier.

Therefore, to achieve consensus over the ordering of end-user requests, we propose to use public blockchain platforms [41,42] for interfacing the federation to the end-users. The consensus algorithms over a public blockchain setting are resistant to sybil attacks. Once the consensus is reached, and an end-user request is committed in the public blockchain, it can be picked up by the federation and all the participants see the same set of requests in the same order. Importantly, users do not need to go through the hassle of registration in a private blockchain, instead they can just create a account (wallet) in the public blockchain and access the federation through the smart contracts.

3.2.2 Brokerless Collaboration

Once the client requests are committed in the federation interface through a public blockchain platform, the federation must handle its scheduling, resource provisioning, access transfer, dispute resolution, etc. In the absence of a trusted cloud broker, all the CSPs must participate using a protocol to carry on all the federation functionalities in a distributed fashion. Again, the decisions taken by the CSPs must represent the decision of the entire federation, as a result the CSPs must come into consensus on the same. Therefore, in this scenario as well, blockchain becomes an obvious choice of a decentralized platform where smart contracts can be used for implementing the collaboration logic and enforcing FLA.

However, public blockchain cannot be used for this task since sensitive information of the CSPs like financial details, credentials, etc. cannot be shared publicly. Also, the collaboration logic like scheduling requests will involve much complex smart contracts involving specific business logic which will be unnecessarily replicated over a public network. Moreover, in order to process a single end-user request, a chain of logic implemented as different smart contracts needs to be executed, which results in a huge latency considering slow transaction commitment times in public blockchains [5,7]. Unlike the *Brokerless Federation Interface* which is to be used by the end-users, the *Brokerless Collaboration* platform will only be used by the CSPs participating in the federation, and hence the participants of the system are known and unlikely to change frequently. As a result, in this scenario, private blockchain can be used which will mitigate the issues of using sensitive information over the blockchain and also provide much faster transaction processing latency.

As a consequence, the *CollabCloud* federation architecture constitutes of two broad platforms as shown in Figure2 – (a) the *Brokerless Federation Interface* developed on a public blockchain platform, and (b) the *Brokerless Collaboration* platform developed over a permissioned ledger.

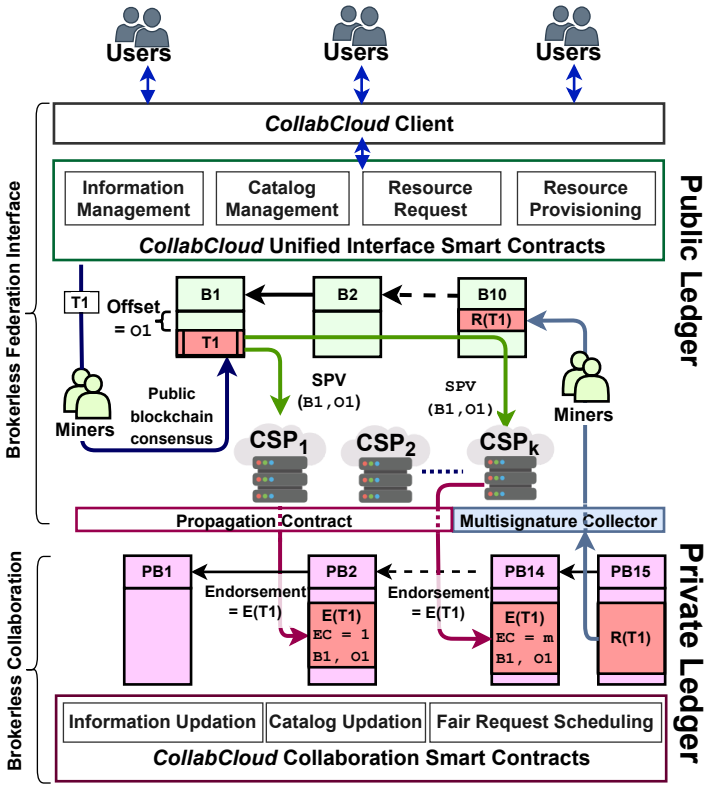


Figure 2: *CollabCloud* architecture overview

3.3 Open Challenges

As we proceed with the design of *CollabCloud*, it is revealed that merely clubbing together any public and private blockchain is not enough; there still exists open challenges that need to be solved, as follows.

(i) **Passing consensus of one network to another:** The public and the private blockchain networks run their own consensus protocols independently. For example, the consensus over the public network should decide the serialization of user requests, and this serialization information would be used by the consensus protocol of the private network for resource allocation. However, in the absence of any central coordinator, it is difficult to securely pass the consensus information from the public network to the private network and vice versa.

(ii) **Verifiability of the federation decision:** The federation's decision of resource allocation comes through a consensus over the private network; but once this information is forwarded to the public network, the cloud

users should be able to verify such decisions to avoid any byzantine behavior from the colluded CSPs.

(iii) **Transferring sensitive information from the closed network to the end users:** Once the federation decides about the resource allocation in the form of VMs, the VM credentials need to be passed to the end-users. Protocols need to be designed to share such sensitive information with the targeted participants.

(iv) **Design of federation functionalities:** In the absence of a broker, the federation functionalities like fair scheduling of requests, resource allocation, etc. need to be designed as smart contracts over the public and private blockchain platforms.

4 Brokerless Federation Interface

CollabCloud provides *Brokerless Federation Interface* over a public blockchain using two components (see Figure2)– (i) ***CollabCloud Client*** which is used by the end-users to interact with the federation, and (ii) ***CollabCloud Unified Interface*** which are smart contracts that are used by both *CollabCloud Client* and the CSPs participating in the federation.

4.1 *CollabCloud* Client

The *CollabCloud Client* is an application which is used by the individual end-users that forms an interface between the users and the *CollabCloud Unified Interface* of the public blockchain. The application is responsible for creating an account (wallet) on the public blockchain and generating a public-private key pair (public key: $\mathcal{P}_{\mathcal{U}}$, private key: $\mathcal{S}_{\mathcal{U}}$). The public key of each user also acts as its identifier and allows the secure transfer of resources through the blockchain (Section 6). Through this client, the end-users can query for information like catalog of resources, pricing, SLA information, and request resources from the federation.

4.2 *CollabCloud* Unified Interface

CollabCloud Unified Interface consists of smart contracts deployed on the public blockchain which are responsible for handling the logic of the unified interface between the end-users and the CSPs. The details follow.

4.2.1 Resource Request Contract

This smart contract allows the end-users to request for resources. A user-request is defined as a four-tuple: $\mathcal{R} = \{\mathcal{R}_{id}, \mathcal{P}_{\mathcal{U}}, \mathcal{V}_j, \mathcal{D}\}$, where $\mathcal{R}_{id}$ is the unique identifier of the user request, $\mathcal{P}_{\mathcal{U}}$ is the public key of the end-user making the request, $\mathcal{V}_j \in \mathbb{C}$ is the VM configuration selected from the catalog $\mathbb{C}$, and $\mathcal{D}$ is the duration for which the VM is requested. This request $\mathcal{R}$ is posted by the end-user through the *CollabCloud Client* which invokes *Resource Request Contract*

through a transaction proposal in a new block for the public blockchain network. The miners mine a new block from the set of transactions posted from the end-users as well as the CSPs. Through open consensus mechanisms, like PoW, PoS, etc. [40] the network as-a-whole agrees on each proposed block that ensures consensus on the transaction data as well as their ordering. Although any business logic can be applied for ordering the requests in a block (and across blocks), in our implementation, we order the transactions based on the decreasing amount of the transaction fee. This transaction fee is an amount proportional to the requested resource capacity, which every user pays against each resource requests[4]; this is a pre-validation token which is adjusted during the billing based on the actual resource usage. Once the user requests are committed in the public blockchain it is uniquely identified by $\mathcal{R}_{id} = \{B, O\}$, where B is the block number, and O is the transaction offset within the block. (see Figure2). Each CSP continuously monitors the public blockchain for a new *Resource Request Contract* transactions through an event listener; once such an event gets triggered, the CSPs collectively transfer the public blockchain consensus it into the federation through *Propagation Contract*, as discussed in Section 5.

4.2.2 User Read-only Contracts

The rest of the smart contracts are read-only for the end-users, where the end-users obtain federation and resource allocation related information. The following contracts are updated by the CSPs through collective consensus via three other smart contracts running over the private blockchain, as discussed in Section 5.

Resource Provisioning Contract allows the CSPs to securely give access of the resources to the end users. The primary objective of this smart contract is to ensure that the allocated resource is an outcome of the collective decision of the federation, and is not from an adversarial CSP. This is particularly important because anyone can try to post an information over the public blockchain because it does not require a pre-authentication of the end-users. A signed encrypted token containing the resource access details are posted on the public blockchain network; the signing procedure is described in details in the *Secure Resource Transfer* sub-module (Section 6.3).

Information Management Contract is used for querying and updating the information of the federation. This includes a list of the CSPs participating in the federation $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2, .., \mathcal{C}_n\}$, and their business information. For example, this should include the name of each participating CSP, their company name, company registration detail, address, contact information, their individual privacy policies etc. Apart from that each CSP will list its public key $\mathcal{P}_{\mathcal{C}_i}$ which also acts as their unique identifier, and the FLA containing the information about the joining date of a CSP in the federation, a copy of the federation joining contract digitally signed by all preceding CSPs and the joining CSP, and

[4]Similar to the credit card validation/pre-authentication for resource requests over popular CSPs, like Amazon Web Services.

the terms to ensure service quality by each such CSP. A transaction to update these is executed by the *Information Updation Contract* within the permissioned DLT of the federation as discussed in Section 5.1.

Catalog Management Contract: is used for updating and querying the catalog of the federation by the CSPs and the end-users respectively. A CSP $\mathcal{C}_i$ can support certain VM configurations which are represented by $\mathbb{V}^{\mathcal{C}_i} = \{\mathcal{V}_1, \mathcal{V}_2, .., \mathcal{V}_m\}$. Each $\mathcal{V}$ is a distinct five tuple: $\{\mathcal{CPU}, \mathcal{MEM}, \mathcal{PS}, \mathcal{LOC}, \mathcal{QOS}\}$, where $\mathcal{CPU}$ is the number of virtual CPU cores in the VM, $\mathcal{MEM}$ is the memory of the VM in GB, $\mathcal{PS}$ is the persistent storage associated with the VM, and $\mathcal{LOC}$ is the location of the data center where the VM will be placed. $\mathcal{QOS}$ represents the quality of service guaranteed by the CSP, such as uptime, and bandwidth. Thus the catalog of the federation is the union of all such VM configurations being offered by the individual CSPs, represented as $\mathbb{C} = \bigcup_{\mathcal{C}_i \in \mathcal{F}} \mathbb{V}^{\mathcal{C}_i}$. A transaction to this contract is executed by the *Catalog Updation Contract* as discussed in Section 5.1.

4.3 Consensus Propagation From the Public Blockchain to the Private Blockchain:

The CSPs of *CollabCloud* must interact with the public blockchain for receiving user requests, as well as the private blockchain for coordinating with other CSPs for scheduling and processing the requests. The most important component in bridging the permissionless *CollabCloud Unified Interface* ledger to the permissioned ledger of the federation is passing the consensus of the public blockchain to the private blockchain. A request received through the public blockchain cannot be processed immediately as (i) an adversarial CSP might start processing a malicious request (ii) CSPs might diverge in agreement over the order of the incoming requests (iii) without agreement on the incoming requests CSPs might duplicate the resource required for serving one request.

The federation should execute the scheduling of a user-request only when it passes through the consensus of the public blockchain, and the CSPs reach consensus on the consensus of the public blockchain (i.e. a particular mined block). This ensures that a malicious CSP cannot collude the federation by triggering the scheduling of an out-of-federation user request or an out of order request.

Consensus on Consensus: The individual CSPs of *CollabCloud* also participate in the public blockchain thereby acting as their own trusted agent. This circumvents the impossibility result [43], stating that cross-chain communication is impossible without a trusted third party. Whenever a new block is committed in the public blockchain, the CSPs in turn it verify it through *Simplified Payment Verification (SPV)* as used in standard public blockchain like Bitcoin [44], and then invokes a *Propagation Contract* in the private federation blockchain network (see transaction T1 in Figure2). The task of the *Propagation Contract* is to collect **verification endorsements** from federation CSPs. The *verification endorsements* are the digitally signed certificates indicating that

the corresponding CSP agrees on the processing of a user-request committed over the public blockchain. Each such *verification endorsement* is committed through the standard BFT protocols [37, 45] of the private blockchain. As a result, a user request can be committed for scheduling if the majority ($> \frac{1}{2}|\mathcal{F}|$) of the federation members endorse the request transaction. The detailed steps are as follows:

Step 1: Endorsement Initialization: Whenever a CSP a "user request" transaction through the event listener of the public blockchain, it checks whether there is already an endorsement available in the private ledger corresponds to that transaction. If no endorsement is available, it initiates the endorsement collection process for that particular request by initiating the `endorsement-count (EC)` variable set to 1, and committing the signed endorsement in the private ledger. The request is also accompanied by the request id $\{B, O\}$.

Step 2: Endorsement Propagation: As other CSPs also get the same request with the same $\{B, O\}$ through the event listener of the public blockchain, they also execute the *Propagation Contract* for it, which adds their signed endorsements while incrementing the `EC`. Each execution of the *Propagation Contract* is also a transaction and hence goes through the consensus process of the private blockchain.

Step 3: Commitment: Thus, the number of endorsements for a request goes up until it reaches greater than half of the number of consortium members ($\texttt{EC} > \frac{1}{2}|\mathcal{F}|$). At this point, the majority of the consortium participants have consensus on the request through endorsements, and each such endorsement has a consensus of the network. Thus, the consumer request is marked as approved and ready to be scheduled.

Theorem 1. The *Consensus on Consensus* mechanism ensures federation interface safety and federation interface liveness.

Proof: Whenever a transaction is committed in a block in the public blockchain, it implies all non-malicious federation CSPs agree on it, along with the request id $\{B, O\}$. The *Consensus on Consensus* mechanism endorses the transactions from the public blockchain and then commits the endorsements in the private blockchain. A transaction is scheduled only when more than $\frac{1}{2}$ of the consortium members endorse the transaction. Given that each endorsement transaction also undergoes consensus in the private blockchain, and the given verifiability property of the private ledger, a transaction from the public blockchain is accepted in the federation only when the majority of the consortium members endorse it. Further, the transactions are executed in the order of $\{B, O\}$ parameters of the public blockchain ensuring agreement on the order. Thus *Consensus on Consensus* ensures federation interface safety.

Consortium interface liveness depends on the liveness of the public blockchain. The event-listeners for correct consortium members eventually trigger the propagation contract when a transaction is committed in the public ledger. Even if there is a temporary fork, the propagation contract is executed when the transaction is finally committed in the public ledger. **End of proof.**

After a user request is accepted in a federation, it is processed through *Brokerless Collaboration* as described in the following section.

5 Brokerless Collaboration

CollabCloud provides *Brokerless Collaboration* over a private blockchain platform using a set of smart contracts for (i) updating information about the federation and the catalog, (ii) fairly scheduling user requests, and (iii) securely transferring the response to end-users.

5.1 Information and Catalog Updation

Information Updation contract on the permissioned ledger enables the participating CSPs to update their own details, as well as the information of the federation as a whole. It also enables new CSPs to join the federation. Whenever a new CSP joins the federation or an existing CSP wants to update its information, it generates a transaction to the private blockchain network which is accepted only if it is able to collect enough endorsements from the other CSPs required for consensus [46]. It can be noted that the joining of a new CSP can be based upon the business logic of the federation. For example, the federation may accept a new CSP only when it conforms to the FLA. In this case, a CSP endorse (sign) the transaction only if the transaction meets the FLA criteria. On successful execution of this contract (upon consensus), it triggers an update to the *Information Management Contract*.

Catalog Updation Contract allows a CSP to update its contribution of resources to the federation. Accordingly, the catalog of the federation is updated in the public blockchain and the contribution of the CSP is evaluated in the private blockchain for fair scheduling of user-requests. Similar to the catalog which is a set of VM configurations, the contribtion of each CSP $\mathcal{C}_i$ is a set of *VM offerings*, denoted by $\mathbb{O}^{\mathcal{C}_i} = \{\mathcal{O}_1, \mathcal{O}_2..\mathcal{O}_m\}$. A *VM offering* is defined as a three-tuple: $\mathcal{O} = \{\mathcal{V}, k, c\}$, where $\mathcal{V}$ denotes a VM configuration as defined previously, k denotes the quantity of the VMs of the particular configuration $\mathcal{C}_i$ can offer, and c denotes the expected pricing of that VM type.

5.2 Fair Resource Scheduling

Once a user request for a VM is committed in the public blockchain, and the same is approved by the *Propagation Contract* of *CollabCloud CSP Interface*, the *Fair Request Scheduling Contract* is triggered. The objective of this contract is to execute a deterministic procedure to decide which among the CSPs should cater to the request, based on the statistics available in the public blockchain through the *Information Updation Contract* and *Catalog Updation Contract*.

CollabCloud design is flexible to allow any scheduling algorithm provided that it is deterministic and the result is verifiable by all the participating CSPs ensuring a consensus in the system. The following constraints ensure this.

C1. All the inputs to the algorithm are based on the information available over either the public blockchain network or the private blockchain network or both.

C2. The output of the algorithm id deterministic.

The above constraints ensure that the output of the algorithm is verifiable by all the participating CSPs as all the CSPs get the same view of the system from the two blockchains.

Algorithm 1: Fair Request Scheduling Contract

Input: $\mathcal{R}_i$, $\mathbb{K}$, $\mathbb{W}$
Result: Scheduled CSP: $\mathcal{C}_s$

for $\mathcal{C}_j \in \mathcal{F}$ **do**
  * Initialize current proportion of scheduled requests of $\mathcal{C}_j$ to 0 *\
  $\mathcal{G}_{\mathcal{C}_j} \leftarrow 0$
  for $l \leftarrow 1$ *to* $|\mathbb{W}|$ **do**
    if $\mathbb{W}[l] = \mathcal{C}_j$ **then**
      $\mathcal{G}_{\mathcal{C}_j} \leftarrow \mathcal{G}_{\mathcal{C}_j} + 1$
    end
  end
  $\mathcal{G}_{\mathcal{C}_j} \leftarrow \frac{\mathcal{G}_{\mathcal{C}_j}}{|\mathbb{W}|}$
  $\mathcal{D}_{\mathcal{C}_j} \leftarrow \mathcal{G}_{\mathcal{C}_j} - \hat{\mathcal{K}_{\mathcal{C}_i}}$
end
$\mathcal{C}_s \leftarrow \text{argmax}_{\mathcal{C}_i \in \mathcal{F}}(\mathcal{D}_{\mathcal{C}_j})$ // Choose CSP with smaller ID in case of a tie.
enqueue($\mathbb{W}$, $\mathcal{C}_s$)
if $|\mathbb{W}| > \mathfrak{K}$ **then**
  dequeue($\mathbb{W}$)
end
return $\mathcal{C}_s$

For *CollabCloud* we propose a *Fair Request Scheduling Contract* (Algorithm 1) which takes into account the contribution of the individual CSPs in the federation and schedules end-user requests in proportion to it. The input to the algorithm is a user request $\mathcal{R}_i$, the proportions of contribution of all CSPs in the federation $\mathbb{K} = \{\hat{\mathcal{K}_{\mathcal{C}_i}} | \ \mathcal{C}_i \in \mathcal{F}\}$, and an array $\mathbb{W}$ consisting of the results of this algorithm for last $|\mathbb{W}|$ scheduled requests. We define infrastructure contribution $\mathcal{K}_{\mathcal{C}_i}$ of each CSP $\mathcal{C}_i$ by the weighted sum of the quantities of its *VM offerings* indicating the amount of IaaS capacity (hardware resources) contributed to the federation by a CSP. The weights can be based on any parameter of the VM configuration; however, for simplicity, here we consider only CPU as the weights, which indicates the number of virtual CPU (vCPU) cores. Therefore,

$$\mathcal{K}_{\mathcal{C}_i} = \sum_{\mathcal{O} \in \mathbb{O}^{\mathcal{C}_i}} \mathcal{O}.\mathcal{V}.\mathcal{CPU} \times \mathcal{O}.k$$

$$\hat{\mathcal{K}_{\mathcal{C}_i}} = \frac{\mathcal{K}_{\mathcal{C}_i}}{\sum_{\mathcal{C}_j \in \mathcal{F}} \mathcal{K}_{\mathcal{C}_j}}$$

Here, $\hat{\mathcal{K}_{\mathcal{C}_i}}$ denotes the proportion of resource contribution by CSP $\mathcal{C}_i$ in the federation. Thus, each time the catalog is updated through *Catalog Updation Contract*, the proportion of contributions are also changed. In order to

schedule the incoming end-user requests in proportion to the contributions, the scheduling contract keeps track of a window ($\mathbb{W}$) of the past scheduled results. We implement $\mathbb{W}$ as a queue containing results for past requests that is $\mathcal{R}_{i-1}, \mathcal{R}_{i-2}, .., \mathcal{R}_{i-|\mathbb{W}|}$. Here each result corresponds to some CSP to which the past request was scheduled. The algorithm first computes the proportion of requests scheduled to a particular CSP as $\mathcal{G}_{\mathcal{C}_j}$, and then computes the proportion deficit as $\mathcal{D}_{\mathcal{C}_j}$. The request $\mathcal{R}_i$ is then scheduled to the CSP having the maximum deficit in its share of past scheduled requests. Then the window $\mathbb{W}$ is updated by inserting the new result, and also removing the oldest result if $|\mathbb{W}| >$ *some threshold* $\mathfrak{K}$.

Theorem 2. Algorithm 1 satisfies constraints **C1** and **C2**.

Proof: Inputs $\mathcal{R}_i$, $\mathbb{K}$, $\mathbb{W}$, are given to the contract algorithm. All the following computations are performed on these values only. End user request $\mathcal{R}_i$ is committed in the public blockchain as well as the private blockchain through consensus propagation. $\mathbb{K}$ is computed by the *Catalog Updation Contract* in the private blockchain. $\mathbb{W}$ is the list of past outputs of Algorithm 1 and hence available from the private blockchain. Therefore, **C1** is satisfied.

In order to show that the output of the algorithm is deterministic we show that each step of the algorithm can be computed deterministically.

For each CSP $\mathcal{C}_j$, the proportion of request scheduled to it is computed from $\mathbb{W}$ as $\mathcal{G}_{\mathcal{C}_j}$. This process is deterministic since input $\mathbb{W}$ is a finite list of CSP IDs. Proportion deficit $\mathcal{D}_{\mathcal{C}_j} \leftarrow \mathcal{G}_{\mathcal{C}_j} - \hat{\mathcal{K}_{\mathcal{C}_i}}$ is computed. $\hat{\mathcal{K}_{\mathcal{C}_i}}$ is obtained from input $\mathbb{K}$, so this step is deterministic.
Finding the CSP with maximum $\mathcal{D}_{\mathcal{C}_j}$ might be non-deterministic in case of two or more CSPs having the same $\mathcal{D}_{\mathcal{C}_j}$, so in such cases we choose the CSP with the lowest ID (example: $\mathcal{C}_2$ in favour of $\mathcal{C}_3$ if $\mathcal{D}_{\mathcal{C}_2} = \mathcal{D}_{\mathcal{C}_3}$).

Therefore, fair scheduling algorithm always produces the output deterministically satisfying **C2**. **End of proof.**

Theorem 3. The output of Algorithm 1 is verifiable by all the correct CSPs in the federation.

Proof: Theorem 2, shows that the output of the algorithm is deterministic. Therefore, to prove that Algorithm 1 is verifiable by all correct CSPs in the federation, it is enough to show that all the correct CSPs agree on the instances of $\mathcal{R}_i$, $\mathbb{K}$, and $\mathbb{W}$ on which the algorithm is executed. For verifiability, it must be ensured that all the CSPs act on the same version of these information. With each execution of *Fair Resource Scheduling* contract, the value of $\mathbb{W}$ is altered. Therefore, the CSPs must know which version of $\mathbb{W}$ is applicable for which transaction. This is ensured in two different ways by *order-execute* based execution and *execute-order* based execution of contracts [46].

Case (i): order-execute – The transactions are ordered first, and there is consensus on this ordering. Then they are executed sequentially, and $\mathbb{W}$ is updated. Thus every CSP applies the transactions in the same sequence on $\mathbb{W}$, starting from the initial version.

Case (ii): execute-order – Each transaction is first simulated on a particular version of $\mathbb{W}$, and this version number is also included in the transaction. Then the simulation result (updated version of $\mathbb{W}$) is sent for consensus. In case of multiple such parallel transactions acting on the same version of $\mathbb{W}$, only one transaction gets consensus and accepted. The rest of them are rejected.

Similarly consensus on the input version of $\mathbb{K}$ maintained even if it is updated through *Catalog Updation Contract*.

Therefore Algorithm 1 maintain verifiability in both *order-execute* and *execute-order* models of smart contract execution. **End of proof.**

6 Secure and Verifiable Information Transfer from Private to Public Blockchain

CollabCloud CSPs reach consensus over federation information, resource catalog, resource scheduling etc., through the private blockchain network. However this private blockchain consensus has no manifestation outside the federation. Hence, end-user clients of the public blockchain network cannot verify the authenticity of the outputs coming from the federation.

In *CollabCloud*, we use the concept of collective signing (CoSi) [9] where a set of authorities collectively sign a valid information to make it verifiable. We utilize *Boneh-Lynn-Shacham* (BLS) cryptosystem [47] for collecting and aggregating signatures from the individual CSPs. Similar to Byzcoin [45] which uses CoSi to reach to a BFT consensus, an information posted in the *CollabCloud* public blockchain is considered to be valid, if and only if it has been signed by at least $\frac{2}{3}$rd of the participating CSPs. The details of the BLS signature computation and verification follow.

6.1 BLS Signatures

A BLS signature is computed as $\sigma_i(\mathcal{M}) = \mathcal{H}(\mathcal{M})^{\mathcal{S}_{\mathcal{C}_i}}$, where $\mathcal{M}$ is the message that is to be signed, $\mathcal{H}(.)$ is cryptographic hash function, and $\mathcal{S}_{\mathcal{C}_i}$ is the secret key of the CSP $\mathcal{C}_i$. The property that makes BLS signatures special is that they can readily be extended to multi-signatures. Therefore, for n CSPs participating in the federation, $\mathcal{C}_1, \mathcal{C}_2, .., \mathcal{C}_n$, the aggregated multi-signature can be calculated as:

$$\begin{aligned} \sigma_{1..n}(\mathcal{M}) = \mathcal{H}(\mathcal{M})^{\mathcal{S}_{\mathcal{C}_1}+\mathcal{S}_{\mathcal{C}_2}+..+\mathcal{S}_{\mathcal{C}_n}} &= \prod_{i=1}^{n} \mathcal{H}(\mathcal{M})^{\mathcal{S}_{\mathcal{C}_i}} \\ = \sigma_1(\mathcal{M}) \times \sigma_2(\mathcal{M}) \times .. \times \sigma_n(\mathcal{M}) &= \prod_{i=1}^{n} \sigma_i(\mathcal{M}) \end{aligned} \tag{1}$$

This aggregated multi-signature $\sigma_{1..n}(\mathcal{M})$ can be verified with the help of the aggregated public key of the individual CSPs. The aggregated public key for n CSPs is computed as $\prod_{i=1}^{n} \mathcal{P}_{\mathcal{C}_i}$, where $\mathcal{P}_{\mathcal{C}_i}$ is the public key of the CSP $\mathcal{C}_i$.

6.2 Posting information using BLS

Any information to be posted in the public blockchain from the federation originates in the form of a transaction from the *CollabCloud Collaboration Contracts*. The federation outputs like updated federation information or updated catalog, etc. must be collectively signed by at least $\frac{2}{3}$rd of the participating CSPs and posted in the public blockchain. This again has two levels of security requirements for two types of information.

(a) Public information (like catalog) to all the end-users, which do not require confidentiality, but require authentication, integrity and non-repudiation.

(b) Private information (like scheduling output) to a particular end-user, which require authentication, integrity, non-repudiation as well as confidentiality.

The *Information Updation Contract* and *Catalog Updation Contract* produce public messages which are meant to be seen by the all end-users. The output of these smart contracts need to be posted in the public blockchain through *Information Management Contract* and *Catalog Management Contract* respectively. The CSP which executes the information update or catalog update transaction obtains the corresponding smart contract output r and constructs a message $\texttt{sign}\{\mathcal{H}(r), \mathbb{B}, [\mathcal{H}(r)]_{\sigma_{\mathbb{B}}}\}$ and forwards it to all the participating CSPs. Here $\mathbb{B}$ is a bitmap indicating which CSPs have signed the message and $[\mathcal{H}(\mathtt{r})]_{\sigma_{\mathbb{B}}}$ is the aggregated signature on the hash of the contract output r. Every CSP, upon receiving this message, adds its own signature through multiplication as shown in Eq. (1), updates $\mathbb{B}$ and sends back the response. Once signatures from requisite numbers of the participants have been aggregated, the final response message $\{r, \mathcal{H}(r), \mathbb{B}, [\mathcal{H}(r)]_{\sigma_{\mathbb{B}}}\}$ is posted in the public blockchain. The authenticity of this message can be easily verified by the using the public keys of the CSPs that have signed the message, and the integrity can be checked by computing and comparing the hash of r. This verification process is carried out by the *CollabCloud Client* which accepts those messages which have the required number of signatures along with the proper hash.

Posting private information through public blockchains requires some mechanism to preserve confidentiality. This is done by encrypting the message using the public key $\mathcal{P}_{\mathcal{U}}$ of the end-user. Thus the final message which is posted in the public blockchain is $\{r, \mathcal{H}(r), \mathbb{B}, [\mathcal{H}(r)]_{\sigma_{\mathbb{B}}}\}_{\mathcal{P}_{\mathcal{U}}}$, where $\{\mathcal{M}\}_{\mathcal{P}_{\mathcal{U}}}$ denotes a message $\mathcal{M}$ encrypted using the key $\mathcal{P}_{\mathcal{U}}$. Thus, only user $\mathcal{U}$ can decrypt the message using its secret key $\mathcal{S}_{\mathcal{U}}$. The decryption and verification of authenticity is handled by the *CollabCloud Client*.

Optimizing the latency for signature collection: The latency of multi-signature mechanism depends on the way the CSPs sign and forward the messages to collectively generate the final signature. following the BFT consensus primitives. One extreme case is when one of the CSPs acts as the leader, and the other CSPs sign their messages and forward it to all other CSPs including the leader (*pre-prepare* phase of PBFT consensus [37]). The leader constructs the collective signature including its own, and broadcasts it among other CSPs

(*prepare* phase of PBFT consensus [37]). Each CSP validates the collective signature, and forwards a commit message to all the CSPs including the leader, indicating the final signature generation (*commit* phase of PBFT consensus [37]). This strategy is likely to have less latency (three rounds of message broadcast), but will have high signature combination computation overhead for one CSP. Another extreme will be to consider a linear chain of CSPs through which the above three rounds of messages propagate; this will have less computation overhead for each CSP but will have high latency. Therefore, similar to the CoSi primitive [9], *CollabCloud* uses a M-ary tree structure to propagate these three rounds of messages through which individual signatures are collected and the multi-signature is constructed following Eq. (1). Interestingly, the latency for multi-signature generation changes with the value of M, which we analyze in Section 9.

Handling denial of service: Off-chain multi-signature collection improves the latency of the process. However, it introduces the risk of denial of service. Although the message to be signed is first committed in the private blockchain through the consensus process, some malicious consortium participants may try to halt the consortium through denial of service attack by not responding to signature collection requests. As a result, to prevent that, *CollabCloud* resorts to a blockchain contract-based signature collection after the off-chain protocol fails (possibly with a timeout). For a message, the *Multisignature Collector* (Figure2) contract is initialized in a similar way as *Propagation Contract*, and gathers BLS signatures of the members. Thus any non-cooperating member is detected through this transparent process.

6.3 Secure Resource Transfer

Although *CollabCloud* enables secure sharing of private information preserving confidentiality, the information are permanently committed in the public blockchain. As a result, sharing infrastructure access tokens such as passwords and keys through the blockchain is insecure, since in a long term those can be compromised and exploited if not regularly changed. Moreover, for transferring IaaS resources, using password-based authentication is insecure and susceptible to brute force attack and man in the middle attack [48]. As a result, *CollabCloud* utilizes the public-private key pairs of the end-users to provide secure access to the provisioned VMs using public key authentication of `SSH Protocol` [8]. Each user request has the user's public key $\mathcal{P}_{\mathcal{U}}$, which is placed in the `authorized_keys` file of `OpenSSH`[5] server configuration in the provisioned VMs. Thus, no sensitive credential has to be transferred across the public blockchain, and the users can access the VM by using its secret key $\mathcal{S}_{\mathcal{U}}$. The IP address of the provisioned VM is shared as a private message as $\{\texttt{IP}, \mathcal{H}(\texttt{IP}), \mathbb{B}, [\mathcal{H}(\texttt{IP})]_{\mathcal{S}_{\mathbb{E}}}\}_{\mathcal{P}_{\mathcal{U}}}$ through the *Resource Provisioning Contract*.

[5] https://www.openssh.com/

7 Processing end-user requests

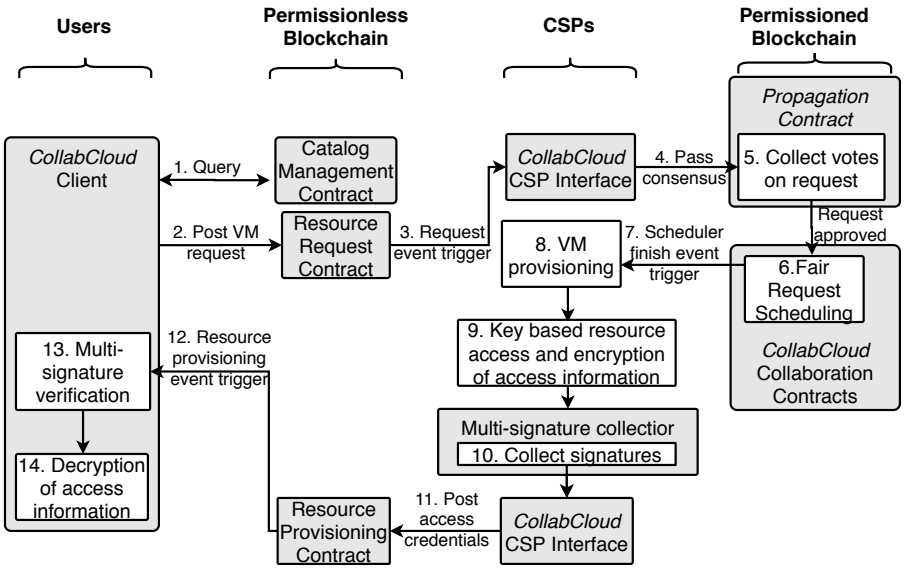


Figure 3: *CollabCloud* system execution flow

Figure 3 shows the end-to-end flow of *CollabCloud* for a resource request from an end-user. After selecting a particular specification from the resource catalog (Step 1), a *CollabCloud* end-user submits its VM request. The *CollabCloud Client* takes this request and executes the *Resource Request* contract on the public blockchain (Step 2). After the request transaction is committed, the CSPs get an event triggered through the *CollabCloud CSP Interface* which starts the *Propagation Contract* for collecting votes on it (Step 4). When the request gets enough votes ($> \frac{1}{2}$ of all CSPs) (Step 6), the scheduling is started in the *CollabCloud Collaboration Contracts* on the private blockchain. Through *Fair Request Scheduling* contract, one CSP is selected (Step 6 and 7), which serves the request. This selected CSP performs VM placement according to its own placement policies (Step 8). Once the VM is provisioned, the CSP enables user access through the secure resource transfer scheme discussed in Subsection 6.3. This involves encryption (Step 9) and multi-signature collection (Step 10) on the resource access information, which is finally committed in the public blockchain through the *Resource Provisioning* contract (Step 11). The *CollabCloud Client* then validates federation CSP signatures (Step 13), decrypts the message (Step 14) and presents the response to the end-user.

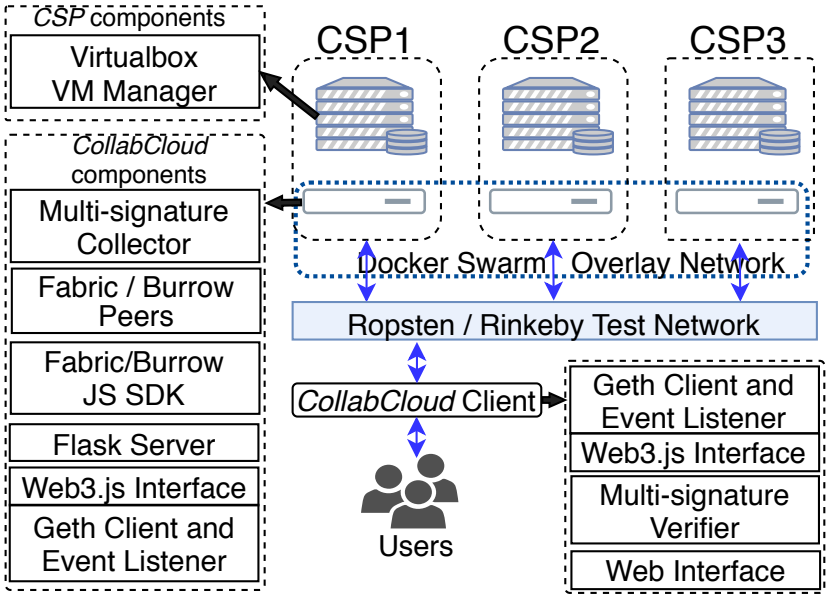


Figure 4: *CollabCloud* modules and Testbed setup

8 Implementation

In order to test the feasibility and practicality of *CollabCloud*, we have implemented each component and deployed the end-to-end system in a testbed (Figure 4). Since *CollabCloud* needs one public blockchain platform for providing *Brokerless Federation Interface*, we have chosen *Ethereum* [41]. For the private blockchain, we have tested with *Hyperledger Fabric* [46] and *Burrow* [6] platforms for *Brokerless Collaboration*.

Starting from the top of the architecture, the *CollabCloud Client* provides a GUI, through which users can query and transact with *CollabCloud Unified Interface*. The frontend GUI is provided by a web interface with `Express.js`[7] server build within the client. The client uses `Web3.js (v1.2.1)`[8] for interaction with Ethereum node. We have used the GO language implementation of Ethereum client, `Geth v1.9.3` [9]. The client also runs an event listener which monitors transactions on Ethereum and identifies any changes in the federation information or response to any user request. In case of federation information update, the same is reflected in the web interface. Any information from the federation is validated by verifying the BLS multi-signature against the public keys of the CSPs. For all BLS signature operations we use the `PBC Library`[10].

[6] `https://www.hyperledger.org/project/hyperledger-burrow`
[7] `https://expressjs.com/`
[8] `https://web3js.readthedocs.io/en/v1.2.1/`
[9] `https://github.com/ethereum/go-ethereum`
[10] `https://crypto.stanford.edu/pbc/`

In the federation side, each CSP has has several components to interact with the public blockchain, participate in the private blockhain network, collect multi-signatures and provision infrastructure as VMs. The interfacing between the end-users and the federation is implemented as a service within each CSP called *CollabCloud CSP Interface* which handles 4-way coordination between (i) the public blockchain, (ii) the private blockchain, (iii) the multi-signature collector, and (iv) the CSP's resource provisioning service. This coordinator is implemented using Python and `Flask`[11] which exposes APIs for interfacing with the other components of *CollabCloud*.

In order to interact with public blockchain (Ethereum), *CollabCloud CSP Interface* provides event listeners for monitoring user requests, and it also lets CSPs to execute transactions on *CollabCloud Unified Interface* smart contracts. The event listeners and smart contract execution is handled by `Web3.js` interacting with `Geth` client.

The interfacing with Hyperledger Fabric is implemented using `Fabric Node.js SDK`. This SDK along with `Express.js` provides the API endpoint implementation. For Hyperledger Burrow, the `Burrow JS SDK` is used along with `Express.js`.

The *Multi-signature Collector* coordinates the collection of BLS short signatures off chain by directly requesting each CSP. In case a CSP goes rogue and does not sign within a given timeout, a *Multisignature Collector* contract for collection of signatures through private blockchain is executed. We use `PBC Library` for signing and verification purposes.

Finally, we had two separate implementations of the *Fair Request Scheduling* smart contract, one for Fabric and another for Burrow. For Fabric, the contract is implemented in *Go* programming language. For Burrow the same is done using *Solidity*[12] language.

9 Evaluation

In order to test the end-to-end functionality and performance of *CollabCloud* architecture along with its various components, we setup a proof of concept (PoC) testbed emulating 3 CSPs participating in the federation.

Table 1: Testbed parameters

Avg network latency between each server	0.28 ms		
Server configurations	CPU	Memory	OS
CollabCloud server	4 Cores (Intel Core i5-4590 @ 3.30GHz)	8GB	Ubuntu 18.04 (Linux 4.15)
CSP server	88 Cores (Intel Xeon Gold 6152 @ 2.10GHz)	256GB	CentOS 7.7 (Linux 3.10)

[11] `https://flask.palletsprojects.com/`

[12] `https://docs.soliditylang.org/en/v0.8.13/`

9.1 Evaluation Setup

Figure 4 shows the setup where each CSP has two cloud servers – one for running the *CollabCloud* services namely, *CollabCloud CSP Interface*, *CollabCloud Collaboration Contracts*, and *Multi-signature Collector*, and the second server for running the CSP's usual services including VM placement and hosting the VMs. All the services are run in `Docker` containers and the networking is established through a Docker swarm overlay network. The configuration of the servers and other testbed parameters are presented in Table 1.

For implementing the CSP IaaS functionalities, we have used `VirtualBox` for creating VMs, and a `Flask` server for accepting VM placement requests and interfacing with `VirtualBox` and *CollabCloud CSP Interface*. Since each CSP has only one emulated data center which is the host server itself, the placement algorithm does not affect the evaluation of our system. However, each CSP has its own set of supported VM specifications, resulting in different catalogs.

In order to evaluate the scalability of different components of *CollabCloud*, we also created a Mininet [49] based network topology for emulation. We created different test scenarios with number of *CollabCloud* CSP nodes ranging from 2 to 32 and the latency between them ranging from 50ms to 400ms in order to capture their performance in real world deployments.

9.2 End-to-end Testbed experiments

In these experiments, we used the PoC testbed setup to evaluate the performance of each component of *CollabCloud* architecture while doing end-to-end user request processing. We used emulated users with numbers ranging from 4 to 64 and programmed them to send parallel requests at the same instance of time. We evaluate the latency and overheads of processing these requests in each *CollabCloud* module.

Brokerless Federation Interface: Each user-request encounters *CollabCloud Unified Interface* twice, hence encountering public blockchain twice, first in the *Resource Request* smart contract and then, in the *Resource Provisioning* smart contract. Figure 5 shows the distribution of total latency (latency in *Resource Request* + latency in *Resource Provisioning* contracts) for processing each user request over two different test networks of Ethereum: *Ropsten*[13] and *Rinkeby*[14]. The processing latency over Ethereum test networks varies widely at different times as these are open blockchain networks, and the processing load varies depending upon the usage by other Ethereum users across the globe. We have collected the data for two weeks at different times of the day, and the same has been plotted in Figure 5. We observe that the PoW-based consensus process of *Ropsten* test network has a higher transaction processing time compared to the PoA-based *Rinkeby* network while executing *CollabCloud Unified Interface* smart contracts; median transaction latency in Ropsten is around 36 seconds, while in case of Rinkeby it is 11 seconds.

[13] `https://ropsten.etherscan.io/`
[14] `https://www.rinkeby.io/`

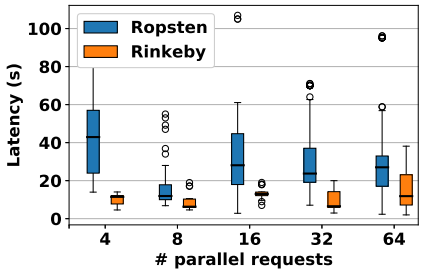


Figure 5: Public Blockchain Latency

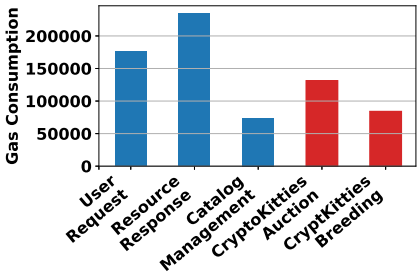


Figure 6: Gas Consumption

Each contract in the public blockchain requires some transaction fees which is proportional to its computational complexity or storage requirements. In Ethereum, this is measured as "Gas". Figure 6 shows the gas consumption of the smart contracts of *CollabCloud Unified Interface*. We observe that *Resource Provisioning* contract is of highest complexity since it has to store the multi-signatures for each transaction, as well as the encrypted resource access information. To understand whether this Gas requirement is too high or too low, we compare these values with respect to the Gas consumption by `CryptoKitties`[15] which is a very popular Ethereum application, and found that they are comparable.

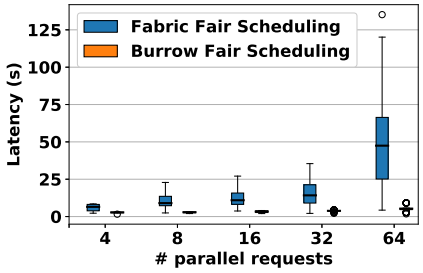


Figure 7: Fair Scheduling Latency: Fabric vs Burrow

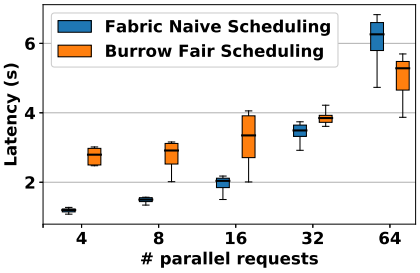


Figure 8: Fabric Static Scheduling vs Burrow Fair Scheduling Latency

Brokerless Collaboration: Figure 7 shows the time required for executing *Fair Request Scheduling* contract in the private blockchain. We observe that the transaction processing time for Fabric is much higher than that of Burrow. The reason for this result is specific to the type of processing required by the *Fair Request Scheduling*. The key difference between Burrow and Fabric is the transaction execution workflow followed by them. Fabric follows *execute-order* flow, while Burrow follows *order-execute*. Executing first and then committing the results introduces a new problem for the type of contracts that read and

[15]https://www.cryptokitties.co/

change the system's common state, just like the *Fair Request Scheduling* uses a history of the already scheduled requests at different CSPs. The reason is as follows. While executing multiple transactions in parallel, let's assume that they get executed on the same current blockchain state $\mathcal{B}_c$, and thus the output is based on $\mathcal{B}_c$. After that, once any one of the transactions is committed, the current state is changed to $\mathcal{B}'_c$. This state change also might change the output of other transactions that would be executed after it. As a result, when the other parallelly executed transactions are processed for committing, they fail in the ordering and validation phase since their execution results do not match with the execution result on $\mathcal{B}'_c$. Fabric does not retry to execute the failed transactions by itself, so *CollabCloud* over Fabric reschedules the failed transactions, thus increasing the latency.

To validate our hypothesis regarding the source of higher overhead caused by Fabric, we also tested with a naive scheduling contract that schedules the requests based on a static rule depending on its ID. This contract does not depend on the current state of the blockchain. In Figure 8, we can see that the scheduling latency of Fabric has dramatically improved. We also noticed that there are no transaction failures due to inconsistent execution results. Moreover, we saw no such latency improvements for Burrow with such a naive scheduling contract. It may be noted that for more parallel requests, Burrow still performs marginally better than Fabric. Consequently, we can conclude that the choice of private blockchain technology depends heavily on the fair scheduling contract's business logic.

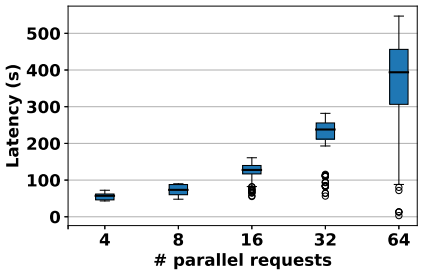


Figure 9: VM Provisioning Latency

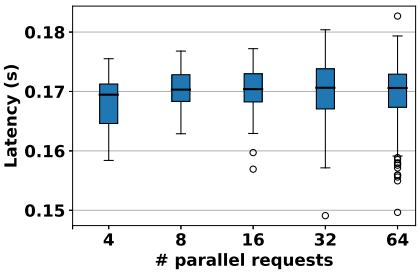


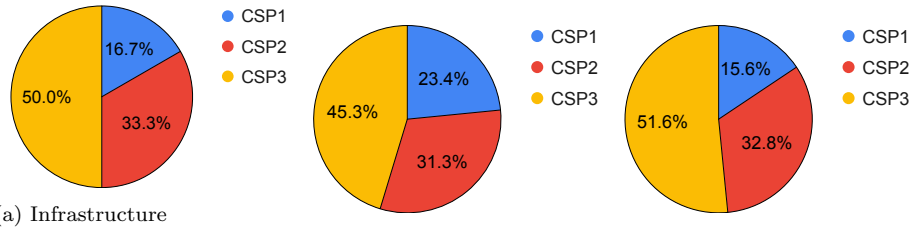
Figure 10: Sign. Collection Latency

After a user request is scheduled, it is provisioned by the selected CSP, and the access information is signed by collecting BLS multi-signatures. Figure 9 shows the distribution of the time taken for VM Provisioning. This increases with the increase in number of parallel requests, mainly due to the limited processing capability of the hardware of our setup. In case of a real CSP, the times will depend on its infrastructure and process of provisioning VMs. Figure 10 shows the distribution of latency for multi-signature collection. We see that the multi-signature collection latency remains fairly consistent with median latency of 170ms.

Evaluation of Fair Resouce Scheduling Algorithm: In order to eval-

Table 2: Catalog and offerings for testbed setup

Catalog					Offering Count		
Configuration	$\mathcal{CPU}$	$\mathcal{MEM}$	$\mathcal{PS}$	$\mathcal{LOC}$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
$\mathcal{V}_1$	1	500 MB	8 GB	L1	6	8	15
$\mathcal{V}_2$	1	800 MB	12 GB	L1	2	8	15
$\mathcal{V}_3$	2	500 MB	8GB	L1	4	8	10
$\mathcal{V}_4$	2	800 MB	12 GB	L1	2	4	5



(a) Infrastructure Contribution (b) Fabric (c) Burrow

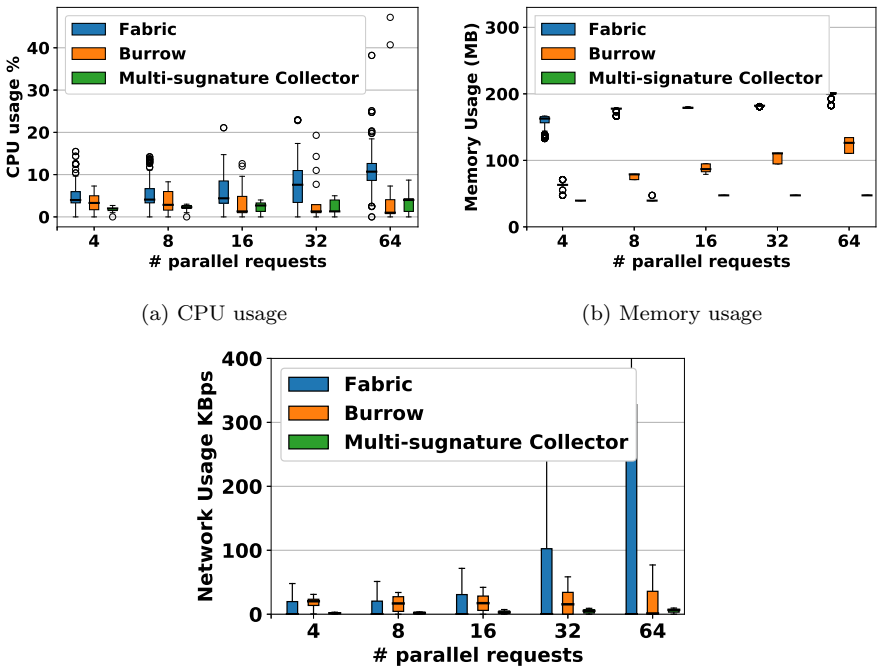
Figure 11: Fair Scheduling Results

uate our *Fair Resource Scheduling* contract (Algorithm 1), we deployed it in our testbed setup as described in Subsection 9.1. The federation offers 4 VM configurations; the catalog and the offerings by the CSPs are presented in Table 2. Figure 11 shows (a) the infrastructure contribution ratio of the CSPs in the federation, as well as the proportion of user requests scheduled to different CSPs through (b) Fabric, and (c) Burrow based implementations. We observe that the proportion of requests scheduled to the three CSPs roughly follow their proportion of infrastructure contribution in the federation indicating fair scheduling.

Resource Consumption: While running all the services of *CollabCloud*, they consume CPU, memory and network bandwidth which are an additional overhead to normal operations of a CSP. Figure 12a shows the distribution of CPU usage by the *CollabCloud* server for executing the private blockchain transactions in Fabric and Burrow, and for multi-signature collection. We observe that the CPU consumption is fairly low, below 10% usage in most cases for all the services. Similarly, Figure 12b depicts the distribution for memory usage, which are below 200MB for all the services. Figure 12c presents mean and standard deviation of the the network utilization. Fabric has the highest network bandwidth consumption among other services, however it is within 300 KBps even for 64 parallel requests.

9.3 Public blockchain scalability

In order to estimate the required transaction throughput in the public blockchain employed in *Brokerless Federation Interface*, we first estimate the rate of VM requests in the most popular cloud service provider, Amazon Web Services (AWS). AWS is estimated to have over a million monthly active users [50]. However, it is unlikely that all users request for new VMs daily. The number of resource



(a) CPU usage

(b) Memory usage

(c) Network Usage

Figure 12: Resource Usage

requests in practice is therefore much less. It is estimated that in 2012, the most popular AWS datacenter was in Virginia, USA, with around 6,000 EC2 VMs [51]. The total number of VMs at that time was about 7,000. Assuming a growth of 20 times from 2012 to 2023 [52], the total number of active VMs now would be 140,000. If even 10% of this total number of VMs are assumed to be created every hour, the number of VM requests per second would be less than 4. As a result, to be comparable with the scale of today's cloud providers, *CollabCloud* has to support a throughput of around 4 VM requests per second. However, the query throughput has to be much higher to facilitate viewing of the catalog and other information about the federation.

Ethereum Query Throughput: Using the *Sepolia* [53] test network through the *Infura* (`https://www.infura.io/`) platform, we achieved a throughput of 3778.57 queries per second. Noticeably, the query throughput increases significantly when multiple queries are batched into a single request. We observed an improvement in query throughput from 1828.15 to 3778.57 queries per second for batch sizes 20 and 400, respectively.

Ethereum Transaction Invocation Throughput: We have tried to introduce as many *CollabCloud VM Request* transactions as possible in a single block to maximize the throughput on the *Sepolia* testnet. We could reliably get

Table 3: Mainnet throughput of different public blockchains

	Block time (sec)	# tx in block	Throughput (per sec)
Ethereum	12	160	13.3
Algorand	3.5	28	8
Solana	0.452	1237	2738
Polygon	2.2	240	109

more than 150 transactions mined in a single block. Although the average block time of Ethereum is 12 seconds, from transaction submission to the block being confirmed, it took $\sim$ 20 seconds on an average in our experiments. Therefore, the VM request throughput achieved was 7.5 per second, which is more than the estimated requirement of 4 per second. Notably, the responses from *CollabCloud* for multiple such VM requests can be batched in a single transaction, thereby not impacting the transaction throughput requirement.

Mainnet Transaction Throughput of Other Public Blockchains: *CollabCloud* architecture is capable of accommodating any public blockchain in the *Brokerless Federation Interface*, as long as the blockchain platform offers a transaction verification protocol similar to SPV of Bitcoin. We compare the transaction throughput of different public blockchains in Table 3. The average block time and throughput values are taken from the main network statistics of the respective blockchains on 19 Dec 2023. Some blockchain platforms however claim a much higher maximum throughput capability. For example, Algorand claims up to $7,500$ transactions per second [42]. Therefore, instantiating *CollabCloud* using public blockchains such as Solana, Polygon, or Algorand can help it achieve the scale required to serve users in the real world.

9.4 Mininet scalability experiments

In this section we focus on the scalability of the closed private network and the multi-signature collection. For this, we setup an experiment with 32 emulated CSPs over a Mininet [49] topology, which form a *CollabCloud* federation. We also changed the inter-CSPs network latency to emulate the CSPs' spread across different geographic regions.

Figure 13 shows the distribution of Burrow implementation of *Propagation Contract*'s execution and consensus latency. The experiment has been done with 2, 4, 16 and 32 CSPs with inter CSP latency varying in each case from 50ms to 400ms. We observe that the median transaction latency lies at around 2.5 seconds with 32 nodes and 400ms inter-CSP latency. Further, the increment in the transaction latency due to an increase in the number of CSPs or inter-CSP network latency is not very high, which indicates the scalability of the proposed approach.

Figure 14 presents the mean and the standard deviation of multi-signature aggregation latency with varying number of nodes and inter-CSP latencies. We observe that the mean latency is below 2 seconds for 16 CSPs and about 3.5 seconds for 32 CSPs. However, the structure of the collection tree can have

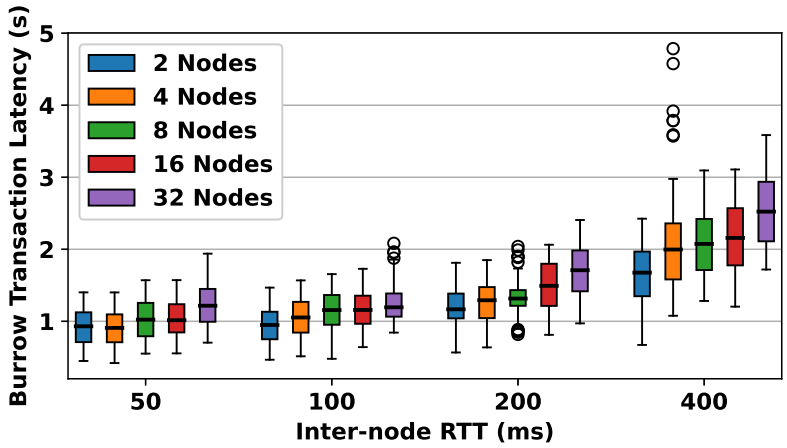


Figure 13: Burrow scalability

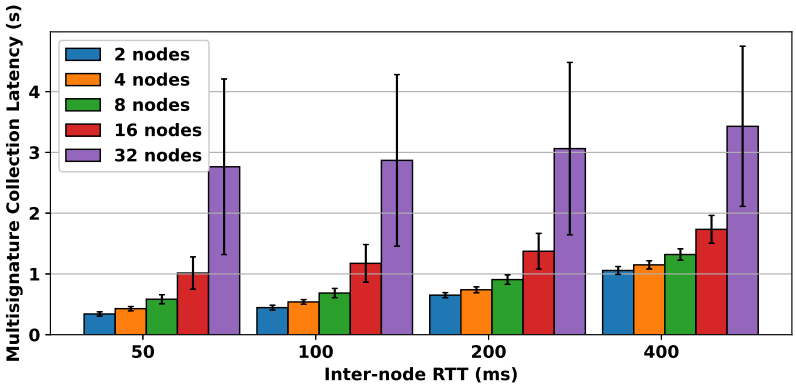


Figure 14: Multisignature scalability

a big impact on the multi-signature collection latency. Thus, we constructed a complete M-ary communication tree with 32 CSPs, with inter CSP latency fixed at 400ms. Table 4 shows the multi-signature collection latency for different values of M. We can observe a sharp improvement in the latency from linear (M=1) to binary tree (M=2) structure. The latency is more or less stable from $M = 4$. However, the multi-signature combination complexity for individual CSPs increase with the value of M. Therefore, the value of M can be chosen based on this trade-off.

Table 4: Effect of communication tree on multi-signature collection latency

M	1	2	4	6	8	16	31
Mean Latency (s)	29.9	5.0	3.2	2.3	2.4	3.0	2.9
Standard Deviation	1.8	0.3	0.2	0.1	0.2	0.6	1.3

10 Discussion

Malicious CSPs not Delivering Promised Resources: In a traditional broker-based federation, if some CSP maliciously avoids delivering the promised resources to the end-users, the users would report this to the broker. Subsequently, the broker would validate the end-user feedback, re-assign the user's request to a different CSP, and take corrective action against the malicious CSP. In the absence of any centralized broker, *CollabCloud* needs a mechanism that is enforced by the consensus of the federation through the private blockchain.

In *CollabCloud*, the end-users can raise a dispute through two special smart contracts. First, the *QoS Dispute Contract* of the public blockchain is used by the end-users to report complaints, which are then ingested into the federation through consensus propagation to the private blockchain. These disputes are then processed by the *Dispute Resolution Contract* in the private ledger. The final resolution result of the federation is also communicated to the end-users through the *QoS Dispute Contract*.

The first and most important task of the *Dispute Resolution Contract* is to validate a complaint. Each CSP in the federation examines a dispute independently, before arriving at the final consensus regarding its validity. We propose using Trusted Execution Environment (TEE) [54, 55] for trustless inspection of the disputed VMs. A TEE provides a hardware-enforced execution environment called an *enclave* that ensures confidentiality and integrity of execution. In addition to the execution output, the TEE provides a *proof of integrity* of the result. Based on a dispute, the CSP that provisioned the VM executes an auditor application in the TEE of the provisioned VM. The results of the auditor application are then committed to the private ledger, where other CSPs can inspect them. The allegedly malicious CSP cannot tamper with this result since the output is accompanied by proof of integrity of execution from the TEE. Moreover, the TEE also provides the signature of the auditor application, which prevents a malicious CSP from replacing it with a tampered application.

The auditor application (a) reports the resources available in the VM (memory, CPU cores, disk space, etc.), (b) runs diagnostics to determine QoS, and (c) examines the access control policies in the VM. A dispute is only accepted as valid if the VM does not meet the specification or QoS goals, or the end-user's access is not granted. Once a dispute is confirmed and agreed upon by the non-malicious CSPs, corrective actions can be taken by re-assigning the end user's request to a new CSP. Further actions can vary depending on the federation policy. We leave the detailed design of the auditor application for future work.

Transferring Payment to the CSPs: The payment from an end-user has to

be processed and passed to the serving CSP when a VM is provisioned. The VM details are communicated to the end-user through the *Resource Provisioning Contract*. The response must also include the public blockchain wallet address of the concerned CSP for payment disbursement. In case of non-payment within a specified time, the CSP shall revoke the VM access and notify the user. A better approach for reimbursement would be atomic swap of a VM access, and its payment. However, this is out of the scope of this work, and we leave it as a future direction.

DoS attacks: A malicious user may flood the system with requests for invalid VM specs that can not be fulfilled. This will prevent other users' requests from being processed, causing a denial of service (DoS). Furthermore, there is an entire low-cost attack vector to evict normal transactions without using high fees [26, 56–58]. Thus, preventing DoS on transaction propagation path from clients to blockchain nodes is still an open research problem in the field, and we leave it for future work.

11 Conclusion

Towards a fully trustless decentralized cloud federation, to the best of our knowledge, *CollabCloud* is the first architecture that removes the federation broker while preserving the unified interface for federation. Through our experiments, we show that the architecture can scale up to at least 32 CSPs with acceptable overheads. Although completely decentralized, *CollabCloud* still needs some mechanisms for the distribution of the public keys of the CSPs, and the distribution of the correct smart contract addresses of *CollabCloud Unified Interface*. Moreover, prevention of DoS attacks is an open challenge. We leave these for future iterations of the system.

References

[1] M. Assis and L. Bittencourt, "A survey on cloud federation architectures: Identifying functional and non-functional properties," *Journal of Network and Computer Applications*, vol. 72, pp. 51 – 71, 2016.

[2] I. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit," *IEEE CLOUD*, pp. 123–130, 2010.

[3] M. Rekik, K. Boukadi, N. Assy, W. Gaaloul, and H. Ben-Abdallah, "Optimal deployment of configurable business processes in cloud federations," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1692–1705, 2018.

[4] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li, "Profit maximization for cloud brokers in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 190–203, 2018.

[5] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[6] B. C. Ghosh, T. Bhartia, S. K. Addya, and S. Chakraborty, "Leveraging public-private blockchain interoperability for closed consortium interfacing," in *IEEE INFOCOM 2021*. IEEE, 2021, pp. 1–10.

[7] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, 2020.

[8] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Protocol Architecture," RFC 4251, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4251.txt

[9] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," in *IEEE Symposium on Security and Privacy*, 2016, pp. 526–545.

[10] H. Li, "Cloud federation as a service," Dec. 30 2014, uS Patent 8,924,569.

[11] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4–1, 2009.

[12] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to enhance cloud architectures to enable cross-federation," in *IEEE CLOUD*. IEEE, 2010, pp. 337–345.

[13] R. N. Calheiros, A. N. Toosi, C. Vecchiola, and R. Buyya, "A coordinator for scaling elastic applications across multiple clouds," *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1350–1362, 2012.

[14] D. G. Kogias, M. G. Xevgenis, and C. Z. Patrikakis, "Cloud federation and the evolution of cloud computing," *Computer*, vol. 49, no. 11, pp. 96–99, 2016.

[15] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–47, 2014.

[16] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing stratos: A cloud broker service," in *IEEE fifth international conference on cloud computing*, 2012, pp. 891–898.

[17] A. Cuomo, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque, and U. Villano, "An sla-based broker for cloud infrastructures," *Journal of grid computing*, vol. 11, no. 1, pp. 1–25, 2013.

[18] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti, "Cloud federations in contrail," in *European Conference on Parallel Processing*. Springer, 2011, pp. 159–168.

[19] G. Zangara, D. Terrana, P. P. Corso, M. Ughetti, and G. Montalbano, "A cloud federation architecture," in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2015, pp. 498–503.

[20] J. Shu, X. Zou, X. Jia, W. Zhang, and R. Xie, "Blockchain-based decentralized public auditing for cloud storage," *IEEE Transactions on Cloud Computing*, 2021.

[21] Y. Zhang, C. Xu, X. Lin, and X. S. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Transactions on Cloud Computing*, 2019.

[22] S. Alansari, F. Paci, A. Margheri, and V. Sassone, "Privacy-preserving access control in cloud federations," in *10th IEEE CLOUD*, June 2017, pp. 757–760.

[23] S. Jiang and J. Wu, "A blockchain-powered data market for multi-user cooperative search," *IEEE Transactions on Network and Service Management*, 2021.

[24] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *IEEE/ACM CCGRID*. IEEE, 2017, pp. 468–477.

[25] X. Ding, J. Guo, D. Li, and W. Wu, "Pricing and budget allocation for iot blockchain with edge computing," *IEEE Transactions on Cloud Computing*, 2022.

[26] Y. Tang, Q. Zou, J. Chen, K. Li, C. A. Kamhoua, K. Kwiat, and L. Njilla, "Chainfs: Blockchain-secured cloud storage," in *IEEE CLOUD*. IEEE, 2018, pp. 987–990.

[27] K. Li, Y. Tang, B. H. Kim, and J. Xu, "Secure consistency verification for untrusted cloud storage by public blockchains," in *SecureComm*, 2019, pp. 39–62.

[28] K. Wang, J. Gao, Q. Wang, J. Zhang, Y. Li, Z. Guan, and Z. Chen, "Hades: Practical decentralized identity with full accountability and fine-grained sybil-resistance," in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 216–228.

[29] B. C. Ghosh, S. Patranabis, D. Vinayagamurthy, V. Ramakrishna, K. Narayanam, and S. Chakraborty, "Private certifier intersection," in *NDSS*, 2023.

[30] F. P. Schiavo, V. Sassone, L. Nicoletti, and A. Margheri, "Faas: Federation-as-a-service," *arXiv preprint arXiv:1612.03937*, 2016.

[31] A. Margheri, M. S. Ferdous, M. Yang, and V. Sassone, "A distributed infrastructure for democratic cloud federations," in *IEEE CLOUD*, 2017, pp. 688–691.

[32] M. Savi, D. Santoro, K. Di Meo, D. Pizzolli, M. Pincheira, R. Giaffreda, S. Cretti, S.-w. Kum, and D. Siracusa, "A blockchain-based brokerage platform for fog computing resource federation," in *Conference on Innovation in Clouds, Internet and Networks*, 2020.

[33] B. C. Ghosh, S. K. Addya, A. Satpathy, S. K. Ghosh, and S. Chakraborty, "Towards a democratic federation for infrastructure service provisioning," in *IEEE International Conference on Services Computing*, 2019, pp. 162–166.

[34] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolić, "XFT: Practical fault tolerance beyond crashes," in *OSDI*, 2016.

[35] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[36] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 203–226.

[37] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[38] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[39] F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic broadcast: From simple message diffusion to byzantine agreement," *Information and Computation*, vol. 118, no. 1, pp. 158–179, 1995.

[40] L. Li, P. Shi, X. Fu, P. Chen, T. Zhong, and J. Kong, "Three-dimensional tradeoffs for consensus algorithms: A review," *IEEE Transactions on Network and Service Management*, 2021.

[41] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[42] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017, pp. 51–68.

[43] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: Communication across distributed ledgers," Cryptology ePrint Archive, 2019, https://eprint.iacr.org/2019/1128.

[44] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," vol. 4, no. 2, p. 15, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[45] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *USENIX Security*, 2016.

[46] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys*, 2018, pp. 1–15.

[47] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT 2001*, 2001.

[48] T. Ylonen, "Ssh key management challenges and requirements," in *IFIP International Conference on New Technologies, Mobility and Security*, June 2019, pp. 1–5.

[49] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[50] A. W. Services, "What is aws? - cloud computing with aws." [Online]. Available: https://aws.amazon.com/what-is-aws/

[51] I. Bermudez, S. Traverso, M. Mellia, and M. Munafo, "Exploring the cloud from passive measurements: The amazon aws case," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 230–234.

[52] U. S. SECURITIES and E. COMMISSION, "Amazon.com, inc. form 10-k. for the fiscal year ended december 31, 2022." [Online]. Available: https://www.sec.gov/Archives/edgar/data/1018724/000101872423000004/amzn-20221231.htm

[53] Ethereum, "Networks — ethereum.org." [Online]. Available: https://ethereum.org/en/developers/docs/networks/

[54] Z. Liu, C. Hu, R. Li, T. Xiang, X. Li, J. Yu, and H. Xia, "A privacy-preserving outsourcing computing scheme based on secure trusted environment," *IEEE Transactions on Cloud Computing*, 2022.

[55] R. Pass, E. Shi, and F. Tramer, "Formal abstractions for attested execution secure processors," in *EUROCRYPT*, 2017, pp. 260–289.

[56] K. Li, Y. Wang, and Y. Tang, "Deter: Denial of ethereum txpool services," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1645–1667.

[57] K. Li, J. Chen, X. Liu, Y. R. Tang, X. Wang, and X. Luo, "As strong as its weakest link: How to break blockchain dapps at rpc service." in *NDSS*, 2021.

[58] Y. Wang, W. Ding, K. Li, and Y. Tang, "Understanding ethereum mempool security under asymmetric dos by symbolic fuzzing," *arXiv preprint arXiv:2312.02642*, 2023.