Towards a Democratic Federation for Infrastructure Service Provisioning

Bishakh Chandra Ghosh*§, Sourav Kanti Addya*§, Anurag Satpathy†, Sandip Chakraborty*, and Soumya K Ghosh*
*Indian Institute of Technology Kharagpur, India; †National Institute of Technology Rourkela, India
Email: {ghoshbishakh, kanti.sourav, anurag.satpathy}@gmail.com, {skg, sandipc}@cse.iitkgp.ac.in

***Abstract*—The current implementations of federated clouds depend on a central broker that takes care of the resource allocation as well as scheduling and pricing for the shared resources under the federation. In this paper, we propose an alternate architecture for federated Infrastructure-as-a-service (IaaS) provisioning with the help of a completely decentralized marketplace designed using the blockchain technology. The proposed architecture is free from any central broker and supports decentralization, transparency of resource exchanges, autonomy of service providers, immutability in information exchange for dispute-free billing and fairness for service provisioning. An in-house implementation of the proposed architecture with three cloud service providers shows that *CloudChain* indeed supports resource allocation fairness while achieving almost similar service provisioning performance compared to a central broker based federation architecture.**

***Keywords*-IaaS; Cloud Federation; Blockchain.**

I. Introduction

In a federated multi-cloud architecture, multiple cloud service providers (CSPs) come in an agreement to share the infrastructure resources, such as computing, memory, storage, etc., among themselves. Such an architecture helps the CSPs to increase the overall revenue and provide better quality of service (QoS) especially during peak loads [1]. Most of the existing federation architectures [2], [3] have the following issues in common. (i) They are managed using a central authority called as *federation broker*. The broker receives service requests from the clients and makes decisions on how to allocate resources from various CSPs to achieve the associated objectives. (ii) The broker handles all the pricing aspects of the CSPs under the federation. Therefore, it is prone to single point failure. (iii) The centralized architecture has its inherent problems like lack of transparency and control, risk of manipulation, possibility of fraudulent activities, etc. The above issues can be tackled by developing a decentralized federation by removing the broker from the architecture. Blockchain [4] based public ledger technology works as a good platform for supporting decentralization. However, a number of research challenges need to be tackled. First, a system needs to be developed to coordinate the federation level agreement (FLA) during execution. Second, the resource allocation scheduling needs to be developed over the decentralized architecture. Third, the infrastructure level services over a cloud, such as virtual machine (VM) resource allocation, VM migration, etc. needs to be coordinated among multiple CSPs.

A few recent works have explored blockchain to support services over a cloud by utilizing public ledger platforms. In [5], the authors have proposed a framework for consumer based data movement policy for clouds by utilizing blockchain. Tosh *et al.* in [6] have proposed a blockchain based data provenance architecture called *BlockCloud* which comprises a proof-of-stake based consensus mechanism for securely capturing the data operations occurring in the cloud environment. Sukhodolskiy and Zapechnikov [7] have built a prototype of multi-user system for access control to data-sets stored in an untrusted cloud environment. However, these works only consider the aspects of a single cloud and are not concerned about the decentralized sharing of resources among multiple clouds. In their recent work, Uriarte *et al.* [8] have proposed a mechanism to manage dynamic service level agreements (SLAs) in a distributed manner with the help of smart contracts. Mainly, they have pointed out the issues and challenges of converting SLAs to smart contracts. In another work, Margheri *et al.* [9] have developed a blockchain based registry approach to ensure proper governance in a federation. Few other recent literature [10] have discussed about blockchain based mechanism for detecting and reporting SLA. However, none of these works mentioned above are concerned about the complete decentralization of the cloud federation keeping all the functionalities of cloud federation intact and holding the interests of the users as well as CSPs.

In contrast to the existing works, we develop a brokerless *decentralized democratic federation marketplace*, called *CloudChain* using a permissioned blockchain model. This democratic architecture eliminates the inherent limitations of any centralized system such as lack of transparency, single point of failure, risk of manipulation, surveillance etc. We consider that the federation involves sharing of resources at the infrastructure level, providing a federated IaaS platform. We implement a prototype of *CloudChain* using open-source Hyperledger Fabric [11] platform and observe that the proposed architecture can provide all the facilities of a decentralized open marketplace with a very marginal compromise on the IaaS performance compared to a centralized broker based architecture, whereas improves system fairness along with other incentives of a decentralized platform.

§ Equal contribution

II. System Architecture

Our proposed federation architecture considers two types of service providers namely, (*i*) demanding CSPs, which suffer from resource limitations under peak loads, and (*ii*) supplying CSPs, that have enough available resources. The demanding CSPs often outsources instantiation requests to the supplying CSPs. Figure 1 gives a broad overview of *CloudChain* architecture, where multiple CSPs share their infrastructure resources with the help of a decentralized exchange developed using the blockchain platform. *Cloud-*

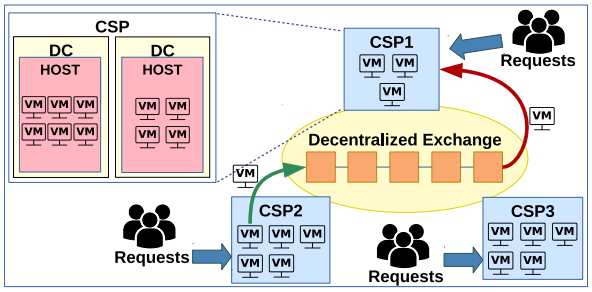

Figure 1. *CloudChain* model overview

Chain consists of multiple CSPs connected over a network that agree to share their idle resources partially or fully. Let $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\}$ be a set of CSPs which provide IaaS and own a cluster of hardware resource units called physical hosts. Together in all hosts, a CSP, $\mathcal{C}_i$ can support certain number of VMs with different VM specifications. Let this be represented by a set $\mathbb{V}^{\mathcal{C}_i} = \{\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_m\}$. Each $\mathcal{V}_j$ is distinct, and its configuration is specified by $\mathcal{V}_j.\mathcal{SP}$, where $\mathcal{SP}$ is the specification of a VM denoted using four tuples – $\{\mathcal{CPU}, \mathcal{MEM}, \mathcal{PS}, \mathcal{LOC}\}$, where $\mathcal{CPU}$ is the number of virtual CPU cores for the VM, $\mathcal{MEM}$ is the memory of the VM in GB, $\mathcal{PS}$ is the persistent storage associated with the VM in GB, and $\mathcal{LOC}$ is the geographic location of the host where the VM is provisioned.

In *CloudChain*, each CSP $\mathcal{C}_i$ offers a subset of $\mathbb{V}^{\mathcal{C}_i}$ with their corresponding specifications and charges. VM request(s) from clients across the globe may come to any $\mathcal{C}_i$ in *CloudChain* with the specification of $\mathcal{R}_{user} = \{\mathcal{SP}, duration\}$, where $\mathcal{SP}$ is the specification for requested VM and $duration$ is the time duration for which the VM needs to be provisioned. According to the requests, a $\mathcal{C}_i$ can either issue the VM resources over its own physical hosts or request some other $\mathcal{C}_j$ in the federation and lease the required infrastructure resources for hosting the requested VM. Therefore, a *CloudChain* CSP can host VMs from both the end-users as well as from other CSPs based on the FLA.

We have taken into consideration both *crash fault* model for handling cases of non-availability or crash of a CSP and *Byzantine fault* model to handle the malicious behavior (for example, denying a provisioned VM or overcharging beyond the provisioning) of the CSPs. In the later, faulty nodes can

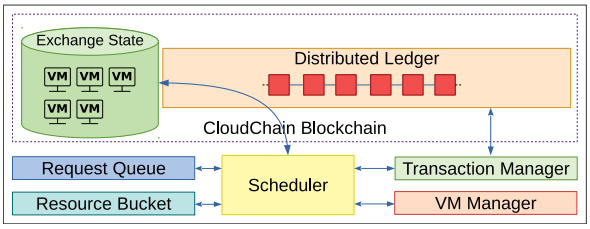

Figure 2. *CloudChain* system components

behave arbitrarily and at most $f = \lfloor \frac{N-1}{3} \rfloor$ nodes can be faulty, where N is the total number of nodes (here CSPs) in the system. Faulty nodes can also try to collude other nodes for compromising the system. However, we assume that the nodes cannot break cryptographic techniques like signatures, encryption, and collision-resistant-hashing.

Different components of *CloudChain*, as shown in Figure 2, are as follows.

(i) **CloudChain Blockchain:** A permissioned blockchain based distributed ledger that forms the core of *CloudChain*. It implements a decentralized exchange where resources are traded between different CSPs with the help of transactions and smart contracts. (ii) **Request Queue:** Each CSP maintains a *request queue* (ReQ) where all the incoming requests for VM provisioning from the clients are queued. (iii) **Resource Bucket:** The resource bucket is a list of available resources, which may include both local resources owned by the concerned CSP or exchange resources which are presently being offered by other CSPs in the *CloudChain Blockchain*. So the resource bucket is further divided into two components: (a) Local resource bucket ($ResB_{local}$) and (b) Exchange resource bucket ($ResB_{excg}$). $ResB_{excg}$ is updated according to the latest state of the *CloudChain Blockchain*. (iv) **VM Manager:** The VM Manager is responsible for creating, destroying and managing VMs in different CSPs. (v) **Scheduler:** The scheduler controls and coordinates different modules of the system. It handles placement of requests and management of resources. (vi) **Transaction Manager:** Transaction manager acts as an interface to the *CloudChain Blockchain*. It receives requests from the *scheduler* when it schedules a request on an remote resource. It also communicates with the *VM manager* to grant access to the resources being offered in the exchange.

Next we discuss the *CloudChain Blockchain* in details, which implements the decentralized marketplace for the exchange of infrastructure resources for IaaS provisioning.

III. CloudChain Blockchain

CloudChain blockchain serves as an information registry that maintains the current state of available resources on offer, pending requests and service agreements. The high level operations that the CSPs can perform on the exchange are – *i)* Offer a new resource for outsourcing, *ii)* Withdraw an

existing offer, *iii)* Query for available resources, *iv)* Request to lease a resource, and *v)* Grant a request.

A. Exchange States

Let $\mathcal{T}$ be the timestamp when an operation, denoted by a unique identity $\mathcal{U}$, is being performed. $\mathcal{I}_S$ and $\mathcal{I}_D$ are the identity of the supplying CSP and the demanding CSP, respectively. $\mathcal{SP}$ is a VM configuration which is being requested by the demanding CSP or offered by the supplying CSP; $duration$ is the time for which a VM is being offered or requested for; $\mathcal{P}$ is the price of an offered VM per unit time. We define *offerings*, *requests*, and *associations* as follows. An **offering** ($\mathcal{O}$) in *CloudChain* is a collection of VM instances which are being offered by the supplying CSPs and can be leased by the demanding CSPs. It is identified by five tuples – $\{\mathcal{U}, \mathcal{T}, \mathcal{I}_S, \mathcal{SP}, \mathcal{P}\}$. A **request** ($\mathcal{R}$) is a solicitation from a demanding CSP for leasing infrastructure service in the form of a VM. It is defined by five tuples – $\{\mathcal{U}, \mathcal{T}, \mathcal{U}_O, \mathcal{I}_D, duration\}$, where $\mathcal{U}_O$ is the identity of the offering that in being requested. An ***association*** ($\mathcal{A}$) denotes an agreement between a supplying CSP and a demanding CSP over a particular *offering*. It is defined using seven tuples – $\{\mathcal{U}, \mathcal{T}, \mathcal{U}_O, \mathcal{I}_D, \mathcal{I}_S, start\_time, duration\}$. Here $\mathcal{U}_O$ is the identity of the offering that in being leased by this association, and $start\_time$ is the starting time of the association.

All the events related to the exchange are recorded in the form of *transactions*. A collection of ordered transactions form a block and each block is linked to the previous block using the hash of that block, forming the blockchain data structure [4]. Verification and total ordering of the transactions are achieved by the consensus protocol. This chain of blocks forms the distributed ledger. Thus, this distributed ledger of transactions is nothing but an immutable history of all the events related to the exchange. From this ledger, any CSP can reconstruct and validate the present state of the exchange, which we call *Exchange State*.

At any time instance i, the *CloudChain Blockchain* maintains an ***Exchange State*** $\mathcal{S}_i = \{\mathbb{O}_i, \mathbb{R}_i, \mathbb{A}_i\}$, where $\mathbb{O}_i = \{\mathcal{O}_{i1}, \mathcal{O}_{i2}, ...\}$ is the set of current *offerings* by the supplying CSPs, $\mathbb{R}_i = \{\mathcal{R}_{i1}, \mathcal{R}_{i2}, ..\}$ is set of current *requests* from the demanding CSPs, and $\mathbb{A}_i = \{\mathcal{A}_{i1}, \mathcal{A}_{i2}, ...\}$ is set of current *associations*. The set of exchange states over time is maintained in the *CloudChain* blockchain, and each CSP maintains its own copy of the blockchain. A new exchange state is included in the blockchain when a transaction is being performed, as discussed next.

B. Transactions

A transaction $\mathcal{T}_i \in \mathbb{T}$, where $\mathbb{T}$ is a set of all possible transactions in *CloudChain*, is defined as a function that operates on a particular state of the exchange to produce a new state. Formally, $\mathcal{T}_i : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{S}$, where $\mathbb{S}$ is the set of exchange states, and $\mathbb{A}$ is a set of all possible operations (offerings, requests or associations). We divide our transactions into two broad categories –

(1) Service Offering Transactions: These transactions are used to add, remove, and manage the *Offerings* on the exchange. These are of two types – (i) *Add-Offering* ($\mathcal{T}_{add}$), (ii) *Remove-Offering* ($\mathcal{T}_{rem}$).

(2) FLA Transactions: These transactions are used to request, and initiate an *agreement* between a supplying CSP and a demanding CSP. These are of two types – (i) *Request-Offering* ($\mathcal{T}_{req}$), (ii) *Grant-Request* ($\mathcal{T}_{grant}$)

Let $\mathcal{S}_i = \{\mathbb{O}_i, \mathbb{R}_i, \mathbb{A}_i\}$ be the latest state of the *CloudChain* blockchain, and $\mathcal{S}_j = \{\mathbb{O}_j, \mathbb{R}_j, \mathbb{A}_j\}$ be the next state after executing a transaction. Then we define different transactions as follows.

Add-Offering takes input a new *offering* ($\mathcal{O}$) and adds it to the set of current *offerings* in the exchange state.

$$\mathcal{T}_{add}(\mathcal{S}_i, \mathcal{O}) = \mathcal{S}_j \ni \mathbb{O}_j = \mathbb{O}_i \cup \{\mathcal{O}\}; \mathbb{A}_j = \mathbb{A}_i; \mathbb{R}_j = \mathbb{R}_i$$

Remove-Offering takes input an *offering* ($\mathcal{O}$) and removes it from the set of current *offerings* from the *Exchange State*.

$$\mathcal{T}_{rem}(\mathcal{S}_i, \mathcal{O}) = \mathcal{S}_j \ni \mathbb{O}_j = (\mathbb{O}_i \setminus \{\mathcal{O}\}); \mathbb{A}_j = \mathbb{A}_i; \mathbb{R}_j = \mathbb{R}_i$$

This transaction is valid only if $\mathcal{O} \in \mathbb{O}_i$, and $\mathcal{O}.\mathcal{I}_S$ is same as that of the CSP executing the transaction.

Request-Offering takes input a *request* ($\mathcal{R}$) for a particular *offering* and adds it to the set of current *requests*.

$$\mathcal{T}_{req}(\mathcal{S}_i, \mathcal{R}) = \mathcal{S}_j \ni \mathbb{O}_j = \mathbb{O}_i; \mathbb{A}_j = \mathbb{A}_i; \mathbb{R}_j = (\mathbb{R}_i \cup \{\mathcal{R}\})$$

The transaction is valid only if the following conditions are satisfied:

(i) The *offering* exists: $\mathcal{O}_k \in \mathbb{O}_i$ such that $\mathcal{O}_k.\mathcal{U} = \mathcal{R}_j.\mathcal{U}_O$

(ii) No other request is already registered against the same *offering* : $\{\mathcal{R}_m \in \mathbb{R}_i \mid \mathcal{R}_m.\mathcal{U}_O = \mathcal{R}.\mathcal{U}_O\} = \phi$

Grant-Offering is executed by the supplying CSP if it grants a *request* ($\mathcal{R}$). It creates an *association* based on the request. The *association* is added to the set of *associations* in the *exchange state*.

$$\mathcal{T}_{grant}(\mathcal{S}_i, \mathcal{R}) = \mathcal{S}_j \ni \mathbb{O}_j = (\mathbb{O}_i \setminus \{\mathcal{O}\}); \mathbb{R}_j = (\mathbb{R}_i \setminus \{\mathcal{R}\}); \mathbb{A}_j = \mathbb{A}_i \cup \{\mathcal{A}\}; \quad \text{where}, \mathcal{O}.\mathcal{U} = \mathcal{R}.\mathcal{U}_O$$
$$\text{and } \mathcal{A} = \{\mathcal{U}, C_T, \mathcal{O}.\mathcal{U}, \mathcal{R}.\mathcal{I}_D, \mathcal{O}.\mathcal{I}_S, C_T, \mathcal{R}.duration\}$$

C_T is the time in the CSP's clock when a transaction is committed. This transaction is valid only if $\mathcal{R} \in \mathcal{S}_i$, and $\mathcal{O}.\mathcal{I}_S$ is same as that of the CSP executing the transaction.

Each of the above transactions can be executed independently by each CSP. A transaction is successfully committed only when all the CSPs agree on it through the consensus process.

IV. Consensus and System Flow

CloudChain needs to ensure that the exchange state in all the CSPs remain consistent, for which we need a consensus mechanism for committing the blocks.

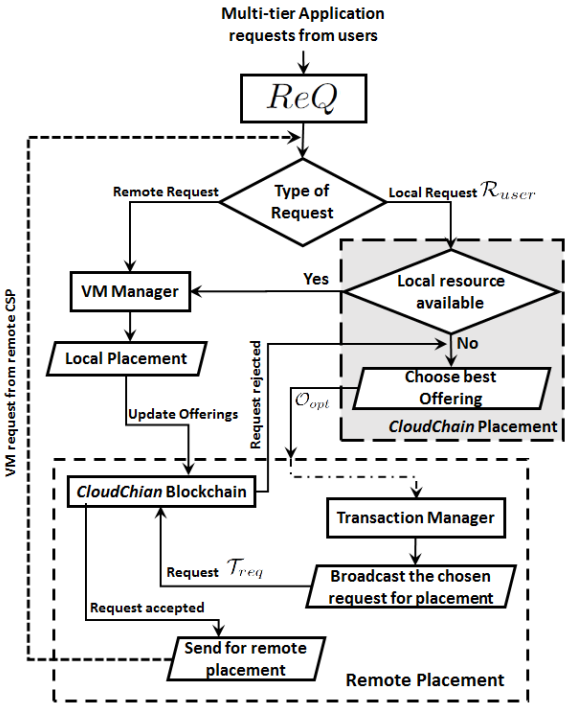

Figure 3. *CloudChain* system flow

A. CloudChain Consensus

The *CloudChain* blockchain is in essence a replicated service with a state, and any CSP can perform a transaction to change the current state, provided that the transaction is a valid one. The system needs to ensure that the transactions are executed in a deterministic way, and the resulting state is safely replicated across all the CSPs. Two important requirements for the correctness of the system are – (a) consensus among the participants on which transactions to execute and (b) agreement on a fair ordering of the transactions.

Since *CloudChain* is based on a permissioned blockchain, we can use any suitable *byzantine fault tolerant* consensus protocol such that all non-faulty CSPs agree upon a valid and fair order of the transactions over a newly proposed block. In order to ensure that client requests are fairly processed, we choose *Redundant Byzantine Fault Tolerance* (RBFT) distributed consensus algorithm [12] which implements a fairness mechanism capable of monitoring a primary (the CSP that proposes a new block) and detecting if it behaves maliciously. Based on RBFT, if the backups observe that the ordering proposed by the master is N edit-distance away (N is a configurable parameter) from their own order, then a view change is requested by electing a new master.

B. CloudChain System Flow

Each CSP participating in *CloudChain* is responsible for serving two kinds of requests – (a) multi-tier application requests from end-users (*Local Request*), and (b) requests for *offerings* from other CSPs (*Remote Request*). The scheduler is responsible for placing the requests. In case of unavailability of local resources, for federated placement, *CloudChain Placement* algorithm finds an optimum compatible *offering*, $\mathcal{O}_{opt}$ from $ResB_{excg}$, based on the service quality and pricing policies. The overall flow of the system from receiving a request to its placement is depicted in Figure 3.

V. Evaluation

We have evaluated *CloudChain* in an in-house testbed setup. The implementation details and obtained results are discussed in this section.

A. Implementation & Testbed Setup

We have implemented a prototype of our proposed framework using Hyperleder Fabric v1.3.0 [11], Docker (https://www.docker.com/), and VirtualBox (https://www.virtualbox.org/). For our in-house testbed set up (Figure 4), we have considered a set of 3 CSPs ($\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$), forming a federation, with each CSP having three emulated data centers (DC). Each CSP is a workstation and to maintain the heterogeneity of resources, we have taken different host configurations for each CSP with $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ having 256 GB, 20 GB, and 8 GB memory respectively. These CSPs are running Linux kernel and capable of spawning VMs using VirtualBox. They are connected through the institute network, and we form a docker swarm constituting these three hosts, and set up an overlay network. Using Hyperledger Fabric, we form three separate organizations with each CSP belonging to a separate organization. Each CSP has one Fabric peer node running in a docker container which is responsible for proposing and committing transactions and maintaining a copy of the blockchain ledger, and one orderer node for participating in the consensus process. We have used the endorsement based consensus mechanism as provided by the Hyperledger Fabric. We have implemented the *CloudChain* blockchain functionalities in the form of chaincodes which are nothing but smart contracts. For each transaction, we implemented appropriate key-level endorsement policies which ensure that required number of signatures must be present for a transaction to be valid.

B. Results

We have compared the VM placement time in three scenarios against a centralized broker based federation. In order to ensure that remote VM placements always take place, we have assumed that each CSP has one data center in a unique location. In each scenario, each of the CSPs get 3 multi-tier applications requests with different location constraints and different number of VMs. In the first, second and third scenarios, each CSP receives 4, 6 and 10 VM requests in total within three multi-tier application requests respectively.

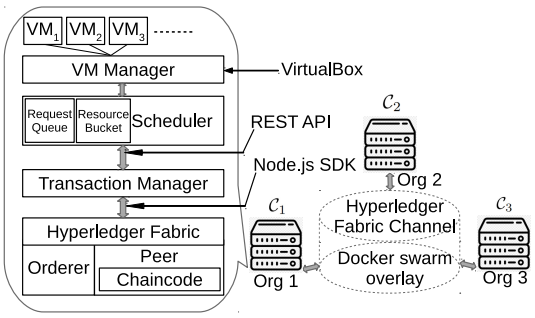

Figure 4. *CloudChain* implementation modules

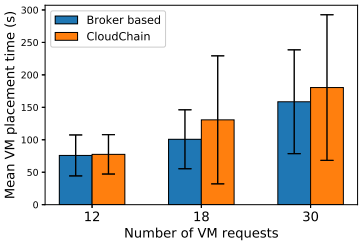

Figure 5. Mean VM placement time

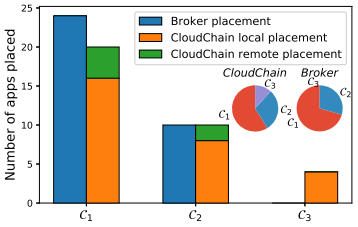

Figure 6. Placement across CSPs

In Figure 5, we observe that the mean placement time and the standard deviation for each VM is very much comparable in both the systems. It is of no doubt that the performance of our proposed decentralized *CloudChain* system is not better than that of a traditional centralized approach in terms of VM placement latency. However, without compromising on the performance, we achieve the benefits of a decentralized system such as transparency, distributed control and fairness.

Next, we have carried out an experiment to justify the fairness properties of *CloudChain* compared to a broker based federation. We have considered 34 parallel multi-tier application requests from end-users, with each application requiring 2 VMs. In *CloudChain*, the number of applications received by the CSPs $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ are 16, 8, and 10 respectively. In broker based system, all the requests arrive at the broker which then handles the placement. Figure 6 shows the distribution of application placements across different CSPs. We observe that in case of the broker based system, most applications are placed in $\mathcal{C}_1$, followed by $\mathcal{C}_2$, and none of the applications are placed in $\mathcal{C}_3$. This violates the fairness property in the federation. In contrast, *CloudChain* provides a much fair distribution of applications as it allows different CSPs to act with autonomy.

VI. Conclusion and Future Work

Towards the goal of removing central controlling authority from a cloud federation and making it completely decentralized, we have proposed a permissioned blockchain based democratic cloud federation architecture called *CloudChain* for IaaS provisioning. From an in-house implementation and testing, we observe that *CloudChain* provides fairness, autonomy, and transparency, with only marginal compromise in performance as compared to a broker based architecture. However, the current implementation of *CloudChain* does not support advanced cloud functionalities, like live VM migration, which we plan to explore as the future direction of this work.

Acknowledgement

We thank Dr. Praveen Jayachandran and Mr. Chander G from IBM Research, India for assistance in implementation with IBM Hyperledger Fabric.

References

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

[2] 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.

[3] 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, Jan 2019.

[4] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *IEEE BigData*, pp. 557–564.

[5] S. Kirkman, "A data movement policy framework for improving trust in the cloud using smart contracts and blockchains," in *IEEE IC2E*, 2018, pp. 270–273.

[6] D. K. Tosh, S. Shetty, P. Foytik, C. A. Kamhoua, and L. Njilla, "Cloudpos: A proof-of-stake consensus design for blockchain integrated cloud," in *IEEE CLOUD*, 2018.

[7] I. Sukhodolskiy and S. Zapechnikov, "A blockchain-based access control system for cloud storage," in *IEEE EIConRus*, 2018.

[8] R. B. Uriarte, R. D. Nicola, and K. Kritikos, "Towards distributed sla management with smart contracts and blockchain," in *IEEE CloudCom*, 2018, pp. 266–271.

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

[10] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao, "A blockchain based witness model for trustworthy cloud service level agreement enforcement," in *IEEE INFOCOM*, 2019.

[11] E. Androulaki. et al., "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in *13th EuroSys*, 2018.

[12] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant byzantine fault tolerance," in *33rd IEEE ICDCS*, 2013, pp. 297–306.