A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement

Huan Zhou[‡†*], Xue Ouyang[†*], Zhijie Ren[§], Jinshu Su[†], Cees de Laat[‡] and Zhiming Zhao[‡]
[‡]Informatics Institute, University of Amsterdam, Amsterdam, 1098 XH, the Netherlands
[†]National University of Defense Technology, Changsha, 410073, China
[§]Department of Intelligent Systems, Delft University of Technology, Delft, 2628 CD, the Netherlands
Email: h.zhou@uva.nl, ouyangxue08@nudt.edu.cn, z.ren@tudelft.nl, sjs@nudt.edu.cn, {delaat, z.zhao}@uva.nl

***Abstract*—Traditional cloud Service Level Agreement (SLA) suffers from lacking a trustworthy platform for automatic enforcement. The emerging blockchain technique brings in an immutable solution for tracking transactions among business partners. However, it is still very challenging to prove the credibility of possible violations in the SLA before recording them onto the blockchain. To tackle this challenge, we propose a witness model using game theory and the smart contract techniques. The proposed model extends the existing service model with a new role called "*witness*" for detecting and reporting service violations. Witnesses gain revenue as an incentive for performing these duties, and the payoff function is carefully designed in a way that trustworthiness is guaranteed: in order to get the maximum profit, the witness has to always tell the truth. This is analyzed and proved through game theory using the Nash equilibrium principle. In addition, an unbiased sortition algorithm is proposed to ensure the randomness of the independent witnesses selection from the decentralized witness pool, to avoid possible unfairness or collusion. An auditing mechanism is also introduced in the paper to detect potential irrational or malicious witnesses. We have prototyped the system leveraging the smart contracts of Ethereum blockchain. Experimental results demonstrate the feasibility of the proposed model and indicate good performance in accordance with the design expectations.**

I. Introduction

Cloud computing is the most popular business model nowadays for sharing resources among multi-tenants. A cloud customer can flexibly use computing, storage, and various other resources from a remote provider as services through the Internet. Such service model conveys great convenience to users but also faces the challenge of "*Cloud Performance Unpredictability*" [1], e.g., when migrating time-critical applications onto clouds [2]. The cloud SLA is an agreement required between a customer and a provider on the quality of certain services; in the case of quality violations, the customer would get the corresponding compensations from the provider.

Traditionally, SLA is a business concept which defines the contractual financial agreements between the roles who are engaging in the business activity [3]. In the context of cloud computing, it is an agreement between the cloud customer and provider on the quality of the cloud service. For instance, the IaaS (Infrastructure-as-a-Service) provider, Amazon Elastic Compute Cloud (EC2), claims that the availability of its data center is no less than 99%. If this number is not achieved, it will pay back 30% credits to its customer as compensation[1]. However, this agreement is hard to be enforced in practice. The major challenges that hinder the conceptual SLA to be feasibly adopted in the real-life industry are:

- **The manual verification:** There lacks an automatic mechanism to enforce SLA agreement, especially for the compensation. EC2[1], e.g., requires customers to claim the SLA violation through emails manually.
- **The fairness between roles:** The provider has more rights in the current model, especially to verify the violation and decide whether to compensate the customer.
- **The proof of violation:** It is hard for the customer to prove and convince the provider that the violation has really happened.

The blockchain [4] technology brings in new opportunities for tackling these challenges. The smart contracts in Ethereum [5] provide a feasible way to automate the service transactions and enforce the SLA via the blockchain. Hiroki et al., [6] introduce a "*Service Performance Monitor*" role to detect the violation and notify the user. However, the proposed solution lacks an analysis of the credibility on the identified violations and still faces challenges in achieving consensus on an event that happens outside the blockchain. The bridge between the events that on and outside the chain is called "*oracle*" [7]. One of the solutions is to retrieve data from oraclize[2], a third trusted company performing as a trustful data source. Moreover, these solutions are centralized, which suffer from single-point of failure and are easy to be compromised.

In this paper, a novel "*witness*" role is introduced to improve the existing blockchain based SLA solutions by detecting violations with explicit credibility concerns in a decentralized manner. The witness is designed as an anonymous participant in the system, who desires to gain revenue through offering the violation reporting service. The payoff function for different actions of the witness in our agreement model is carefully designed in a way that the witness has to always behave honestly in order to gain the maximum profit for itself. To be specific, the trust issue of the witness in our model is proved by game theory by using the Nash equilibrium principle.

*The first two authors equally contributed to the paper.

[1]https://aws.amazon.com/compute/sla/
[2]http://www.oraclize.it

In addition, an unbiased random sortition algorithm is developed in our witness model to select a certain number (pre-defined through negotiation between the service provider and the customer) of witnesses in order to form a committee. The committee members are randomly selected and the randomness cannot be dominated by any participant. This is very important to avoid situations when the majority of the delegates are representing the same side: either the customer or the provider, to ensure fairness. Moreover, this algorithm is also capable of eliminating the opportunity of collusion because the committee members are not pre-determined, and there is no chance for them to know each other in advance. Last but not least, a prototype system[3] using smart contracts of the Ethereum blockchain is implemented to automate the SLA lifecycle and empower the fairness between roles, especially for the customer. The experiment is conducted on Rinkeby[4], which is a world-wide blockchain test net for developers to debug the developed smart contracts. The experimental study demonstrates the feasibility of our witness model and the system performance.

The rest of the paper is organized as follows. Section 2 discusses the related work on cloud SLA and blockchain; Section 3 presents the witness model design with smart contracts; Section 4 details key techniques including the unbiased random sortition algorithm and the payoff function design, the trustworthiness of which is proved by the Nash equilibrium principle; Section 5 introduces the prototype implementation and experiments; Section 6 summaries the paper with conclusion and future work.

II. Related Work

SLA is a well-discussed research topic, specifically in the context of cloud computing. It establishes the quality of service agreement between the service provider and the customer, which ensures the customer's benefit when the agreement is violated. A typical SLA lifecycle consists of multiple enforcement phases, including negotiation, establishment, monitoring, violation reporting and termination [3]. Most of the research work focus on three aspects: (1) syntax definition of the SLA terms and parameters, the goal of which is to standardize the representation so that SLA can be easily processed online by computer systems; (2) resource allocation techniques to ensure the SLA. The work in this aspect focuses on the algorithm to optimize resource allocation, thereby avoiding SLA violation from happening. SLA in this kind of work is typically considered as constraints; (3) systems or methods to address issues in specific phases of the SLA lifecycle. According to a systematic survey [3] on cloud SLA, among this kind of works, 22% are focusing on negotiation and establishment phase, 73% are targeting at monitoring and deployment phase, 3% are interested in SLA violation management while 1% focus on reporting. For these phases, the goal of negotiation is to maximize either the provider's or the customer's revenue through adopting some negotiation strategy [8]. Monitoring is mainly about what to monitor and how to automate the process [9]. However, the most challenging phases, violation management and reporting, are seldom explored. In industry, Amazon CloudWatch service[5] is an example that the provider automates monitoring and notification. In this case, the customer has no choice but to trust the provider. Muller [10] develops a platform named SALMonADA to deal with the SLA violation at runtime. It works as a third trusted party to perform the monitoring and violation reporting. All these work assume that the violation reporting is trustworthy, which is, however, the most difficult part in reality.

Smart contract is proposed to digitally facilitate, verify and enforce a contract through a computer protocol [11]. Some explorations, e.g., [12], combine this concept with cloud SLA negotiation, focusing on the semantic expression of smart contract to automate the negotiation phase. However, most of them lack a trustworthy platform to execute the smart contract. This is actually important because the smart contract relies on a strong assumption that no one can tamper its execution. Town Crier [13] and TLS-N [14] ensure the trustworthy execution and communication environment from the hardware and transmission protocol level, respectively. However, they are either centralized or require special infrastructure support.

Blockchain [4] is a promising technique to be used as the execution platform: interactions on the chain are immutable, therefore it can ensure the trustworthiness required by the smart contract. Ethereum [5] first realizes to execute a general-purpose program on its blockchain. Hiroki et al., [6] leverages Ethereum and designs a set of web APIs to automate the SLA lifecycle enforcement on the blockchain. A new role called "*Service Performance Monitor*" is introduced in their system, which is responsible for the violation reporting. However, it is not discussed in the paper whether the violation reports sent to the blockchain can be trusted. Actually, this is still a gap for blockchain based systems: how to credibly record a random event onto the chain when the event happens outside the blockchain. Currently, one of the dominant solutions is using "*oracle*" [7] to fill this gap, an agent performing as a "data-carrier" for the blockchain. Oraclize[2] is a trusted company acting as the third party, offering the service as an oracle. But it exists single-point of failure and deviates from the decentralization idea of blockchain. To cope with this, ChainLink [15] works on distributed oracles. Distributed oracles act in a way that, only when an agreement is achieved among the oracles, the result data of the event can be carried onto the chain or can trigger a transaction. This idea still faces some downsides, such as no incentive for individuals to do the duty, requiring individuals to be independent and trustworthy, the consensus issue among different oracles, etc.

III. The Witness Model using Smart Contract

The roles involved in the proposed model are introduced in this section, specifically, the role of the witness. The overall

[3]https://github.com/zh9314/SmartContract4SLA

[4]https://www.rinkeby.io/

[5]https://aws.amazon.com/cloudwatch/

system architecture for SLA enforcement using the smart contract on blockchain is illustrated afterwards, followed by a detailed description of the responsibility of the witness: service violation detection and reporting.

A. The Witness Role and the Notations

There are mainly two roles in the traditional cloud SLA lifecycle: the cloud provider, P, which offers cloud service; the cloud customer, C, which consumes the cloud service and pays the service fee. We formulate the scenario as follows using a basic example to demonstrate our work.

In this paper, we assume an IaaS provider p providing VMs (Virtual Machine) on demand according to the request of its customer c with a public address IP_{pub}. During the service time, $T_{service}$, only c can SSH and login to the assigned VM through the corresponding address IP_{pub}. In this case, SLA can make a commitment that, during $T_{service}$, the provisioned VM is always accessible. If this is true, c must pay the service fee, $F_{service}$, to p after the service ends. However, in the case of a SLA violation (the commitment is not realized), c must acquire a compensation fee, $F_{compensation}$. In other words, c only needs to pay $F_{service} - F_{compensation}$ to p in the end, where we assume that $F_{service} > F_{compensation}$. In addition, the case that the inaccessibility is caused by the customer's own network problem is excluded from being classified as a SLA violation event.

In order to convince both roles whether the violation indeed happens, we bring in another new role into the traditional SLA lifecycle, named as witness W, and leverage the blockchain to play as the trusted party to afford a platform for these roles and enforce these monetary transmissions. The witnesses are also normal participants in the blockchain and volunteers to take part in the SLA system to gain their own revenue through offering monitoring service. In order to solve the trust issue, a witness committee is selected to participate a specific SLA lifecycle, consisting of N witnesses, $\{w_1, w_2, ..., w_N\}$. They together report the violation event, and obtain witness fee $F_{witness}$ as rewards from both the provider and the customer. The wallet address of a specific role on the blockchain is denoted as $.address$. For instance, $w_k.address$ is the wallet address of witness w_k.

In this paper, a basic assumption is made on the witness role, that they are always selfish and aiming at maximizing its own revenue.

B. Overall System Architecture

Figure 1 illustrates the overall architecture we design for cloud SLA enforcement. The proposed system consists of two types of smart contracts based on blockchain: the smart contract of a witness pool, which is the fundamental smart contract of the system; the smart contract of a specific SLA, which is for the SLA enforcement. For the witness-pool smart contract, there are mainly three responsibilities, including witness management, specific SLA contract generation and witness committee sortition. Any user of the blockchain, who has a wallet address, can register its wallet address in the witness pool to be a member of witnesses. They can keep themselves online and wait to be selected for some specific SLA contract. The incentive for the witness to participant in this system is to obtain revenue. And the more witness participant in the system, the more reliable and trustful the system would be.

The entire SLA lifecycle under our system then becomes as follows. Certain provider p first leverages the smart contract of the witness pool to generate a SLA smart contract for itself. Prior to setting up SLA, the customer c should negotiate with the provider p about the detailed SLA terms, including $T_{service}$, $F_{service}$, $F_{compensation}$, and etc. Thereinto, one of the most important terms is to determine N, the number of witnesses that would be hired for enforcing this SLA. The more witnesses involved in a SLA, the more trustworthy the violation detection results would be. On the other hand, however, the more witness fee would be paid and both the customer and the provider need to afford this fee equally. According to the result of negotiation, the provider is able to customize these parameters and generate a new SLA smart contract. Afterwards, a set of N witness members can be selected to form a witness committee through the sortition algorithm detailed in Section IV-A, which is also implemented by the witness-pool smart contract on the blockchain. The algorithm is designed to be unbiased and random, guarantees that witnesses selected in the committee are independent and would not belong to a specific side, either c or p, achieving mutual trust from both roles. Meanwhile, the provider provisions its cloud service for the customer to use and is able to publish its service detail in the SLA smart contract. The witnesses committee is therefore notified to start monitoring the service according to these details. Using previous IaaS case as an example, the provider provisions a VM on demand and inform all the committee members with the public address

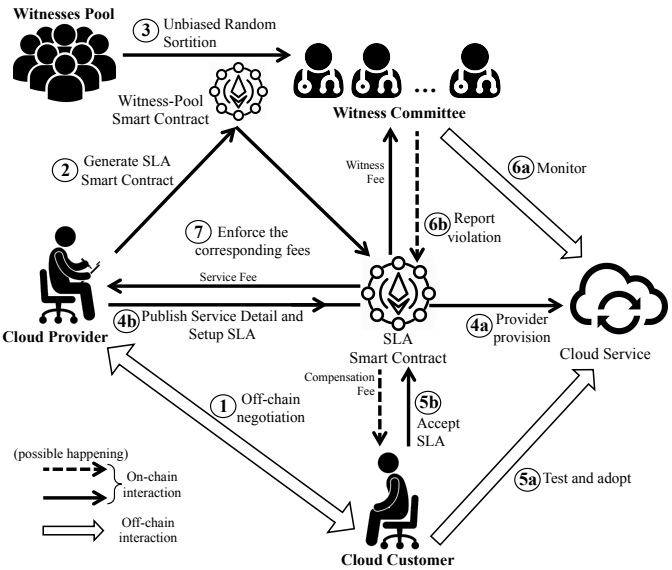

Fig. 1: System Overview for Cloud SLA Enforcement

IP_{pub} and the customer through the service detail field of the SLA smart contract. Not only the customer can use the VM, each witness is also able to "ping" the address IP_{pub} constantly. If the violation happens during the service time, i.e. the address IP_{pub} is not accessible, the witness can report this event immediately.

Since the first violation report, the smart contract would start counting a time window, T_{report}. Within this time window, the smart contract accept reports from other witnesses. When the time window T_{report} is over, the violation is automatically confirmed, if there are no less than M out of N reports from the witness committee received by the smart contract. M is also negotiated by p and c. It is then defined in the SLA smart contract. Of course, M must be bigger than the half of N. Furthermore, the bigger the M is, the more trustworthy the violation confirmation is. For example, if there are $N = 3$ witnesses in the committee, the service violation can only be confirmed when at least $M = 2$ of them report the event. Here, it is worth to mention that the smart contract is designed to receive the report only from the committee member. And the second report from the same witness is refused within the same report time window. In some sense, these N independent witnesses constitute a $n\text{-}player$ game, in which each witness would like to maximize its own revenue. We specially design the payoff function, shown in Section IV-B, and leverage the Nash equilibrium principle of game theory to prove that the witness has to be a honest player in this game. That is they have to report the violation according to the real event.

Finally, the SLA ends at two cases. One case is the service time $T_{service}$ is over and there is no violation. The other case is that the SLA is violated. According to these different cases, the three roles are able to withdraw corresponding fees from the SLA smart contract. This is explained in detail in Section IV-B.

IV. Key Techniques

In this section, we describe key techniques adopted in our witness model in detail. This model enables the automatic detection on the SLA violation, the results of which can convince both sides: the provider and customer. First, the unbiased random sortition algorithm is leveraged to guarantee that most of witnesses selected into the committee are random and independent. It is also important to make both sides achieve consensus that most of the selected witnesses would not delegate the opponent's benefit. Based on this, we give the payoff function for the witness model. And also through the Nash equilibrium principle, we prove that the "player" from the witness committee has to behave honestly and tells the truth to maximize its revenue. Furthermore, we analyze some possible fraudulent behaviors from a malicious witness and demonstrate it can be audited through its action history.

A. Unbiased Random Sortition

It is crucial in the witness model that the witness sortition for a specific SLA contract is unbiased, i.e., neither the provider nor the customer can have advantages in the committee selection. Since Ethereum has already been introduced as a trusted party in our model, we propose a straightforward random sortition algorithm for committee selection shown as Algorithm 1, which is also implemented in the witness-pool smart contract. It is different from our previous algorithm, whose randomness comes from participants [16].

Firstly, there is a basic smart contract to manage the witness pool. It affords a set of interfaces for any blockchain user to register into the pool. Moreover, the registered witness is able to turn its state to "Online" or "Offline", in order to indicate when it can be selected. The detailed witness state management is shown in Section V-A. The addresses in the witness pool are managed as a list in the registration order.

There are two interfaces designed in the smart contract to select N witnesses from the pool. The "request" interface is firstly invoked by a specific SLA smart contract at block B_b. It means this transaction is involved in the b^{th} index of block. The hash value of this block is B_b^{hash}. After K blocks generated by the blockchain, another interface "sortition" can be invoked to select N online witnesses as candidates. The

Algorithm 1 Unbiased Random Sortition

Input:
Registered witness set, RW, a list of addresses;
The the size of the list, $len(RW)$;
The number of online witnesses, oc;
Required number, N, of members in a witness committee;
The hash value, B_b^{hash}, of the b^{th} block B_b at request;
The block index, Id, of current block;
Following sequential, K_s, blocks;
Confirmation, K_c, blocks;
The address of the provider, $p.address$;
The address of the customer, $c.address$

Output:
Selected witness set, SW, to form a committee.

assert($Id > b + K_s + K_c$) && assert($oc >= 10 * N$)
seed $\leftarrow 0$
for all $i = 0$; $i < K_s$; $i++$ **do**
  seed$+= B_{b+1+i}^{hash}$
end for
$SW \leftarrow \emptyset$
$j \leftarrow 0$
while $j < N$ **do**
  $index \leftarrow$ seed$\%len(RW)$
  if $RW[index].state == Online$
  && $RW[index].reputation > 0$
  && $RW[index].address != c.address$
  && $RW[index].address != p.address$ **then**
    $RW[index].state \leftarrow Candidate$
    $oc--$
    Add $RW[index] \Rightarrow SW$
    $j++$
  end if
  seed $\leftarrow hash$(seed)
end while
return SW

sortition algorithm is shown in Algorithm 1.

It takes the hash values of the former K_s out of K blocks mentioned above as a seed. In addition, we need to wait other K_c blocks to confirm the adopted former ones, where $K_s + K_c = K$.

- Here, K_s should be chosen such that the probability of some parties sequentially generating K_s blocks is very small.
- K_c needs to be chosen such that the candidate blocks before are finally involved in the main chain with a dominant probability.
- These two values depend on the blockchain's own properties. Considering the main net of Ethereum, [17] shows that the top four miners control 61% of the mining power. Thus, we recommend $K_s = 10$ so that with more than 99% probability that the seed is not manipulatable and predictable even if the top four miners collude. On the other hand, it is commonly believed that K_c should be 12.

Only the "Online" witness with a positive reputation can be selected by the seed from the list of witness address pool. Anyhow, a new seed is generated based on the hash value of the previous seed. This process is repeated until required N witnesses are selected. In the beginning of Algorithm 1, it is firstly checked that there are at least 10 times of available witnesses than required N, which ensures the output of the algorithm as well as randomness.

Considering the difficulty itself of generating hash value for a block and combining the sequential K blocks as seed, we can prove that the sortition algorithm is random and unbiased, i.e., neither the provider nor the customer can manipulate the sortition result to take advantages in the committee.

B. Payoff Function and Nash Equilibrium

Game theory targets to mathematically predict and capture behavior in a strategic situation, where each player's revenue depends on the strategies of itself and also others. There is currently a wide range of applications, including economics, evolutionary biology, computer science, political science and philosophy [18].

The strategic or matrix form, of a *n-player* game, is the most common representation of strategic interactions in game theory. The definition consists of a set of players, a set of strategy profiles and a design of payoff functions. Based on the basic type of strategic form game with complete information in game theory, we define our witness game as follow.

Definition 1. Witness Game: It is a *n-player* game represented as a triple (SW, Σ, Π), where

- $SW = \{w_1, w_2, ..., w_n\}$ is a set of n players. Each player is a selected witness and they form the witness committee.
- $\Sigma = \Sigma_1 \times \Sigma_2 \times ... \times \Sigma_n$ is a set of strategy profiles, where Σ_k is a set of actions for the witness w_k, i.e., w_k can choose any action $\sigma_k \in \Sigma_k$. A strategy profile is therefore a vector, $\sigma^* = (\sigma_1^*, \sigma_2^*, ..., \sigma_n^*)$, where σ_k^* is a specific action of Σ_k, $(k = 1, 2, .., n)$.
- $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$ is a set of payoff functions, where $\pi_k : \Sigma \rightarrow R$ is the payoff function determining the revenue for witness w_k under a certain strategy, $(k = 1, 2, .., n)$. R is the corresponding revenue.

In addition, $\sigma_{-k} = \{\sigma_1, \sigma_2, ..., \sigma_{k-1}, \sigma_{k+1}, ..., \sigma_n\}$ is defined as any strategy profile σ without player k's action. The full strategy can then be written as $\sigma = \{\sigma_k, \sigma_{-k}\}$. Actually, there are only two actions in our witness game, which is $\Sigma_k = \{\sigma_k^{(r)}, \sigma_k^{(s)}\}$. $\sigma_k^{(r)}$ means Report the service violation to the smart contract. $\sigma_k^{(s)}$ means do not report and keep Silence to the smart contract. In this N-witness game, we define the set of witnesses choosing the action of Report as, W_{report}, where $\forall w_k \in W_{report}$, the $\sigma_k^* = \sigma_k^{(r)}$. Respectively, $W_{silence}$ is the set of witnesses not reporting, where $\forall w_k \in W_{silence}$, the $\sigma_k^* = \sigma_k^{(s)}$. These actions determine the final state of SLA: $SLA_{status} = V$iolated, there is a service violation; $SLA_{status} = C$ompleted, service is completed without violation. We then define the violation confirmation as Definition 2.

Definition 2. Violation Confirmation: Based on the result of a strategy profile in a N-witness game, the service violation is confirmed, only when $\|W_{report}\| \geq M$, where $1 < N/2 < M < N, N, M \in \mathbb{N}$. Otherwise, it is treated as there is no violation happened.

It is worth mentioning that we define $N > 2$ and $M < N$ here, in order to achieve the violation confirmation reliably and fairly. According to our witness model, the witness is designed to report the violation along with endorsement fee to the SLA smart contract. Therefore, if the violation is not confirmed, the witness cannot retrieve back its endorsement fee as a penalty. The detailed payoff function design is shown as Definition 3 according to above definitions and analysis. Thereinto, the value of each function is only leveraged to quantitatively represent the relative relationship among the fees. Hence, 1 represents one share of profit. 10 is ten times shares of 1. -1 means losing one share of profit.

Definition 3. Payoff Functions: The values of payoff functions are designed according to the final SLA status.

- When $SLA_{status} = V$iolated:
  $\forall w_k \in W_{report}, \pi_k(\sigma_k^{(r)}, \sigma_{-k}) = 10$;
  $\forall w_k \in W_{silence}, \pi_k(\sigma_k^{(s)}, \sigma_{-k}) = 0$.
- When $SLA_{status} = C$ompleted:
  $\forall w_k \in W_{report}, \pi_k(\sigma_k^{(r)}, \sigma_{-k}) = -1$;
  $\forall w_k \in W_{silence}, \pi_k(\sigma_k^{(s)}, \sigma_{-k}) = 1$

In a *n-player* game, if a player knows the others' actions, it would choose a strategy to maximize its payoff. This is referred as its best response. Therefore, the best response of the witness w_k can be defined as follows.

Definition 4. Witness w_k's best response: In order to maximize its revenue, w_k's best response to a strategy profile σ_{-k}^*

is a strategy $\sigma_k^* \in \Sigma_k$, such that $\pi_k(\sigma_k^*, \sigma_{-k}^*) \geq \pi_k(\sigma_k, \sigma_{-k}^*)$ for $\forall \sigma_k \in \Sigma_k (k = 1, 2, ..., n)$.

A Nash equilibrium point [19] is therefore able to be defined as a stable state, where no player has an incentive to deviate from current strategy. It is actually a strategy under which every player adopts its own best response.

Definition 5. Nash equilibrium point: It is a specific strategy profile $\sigma^* = (\sigma_k^*, \sigma_{-k}^*)$, if for every witness w_k, σ_k^* is a best response to σ_{-k}^*, i.e., $\forall w_k \in SW$ and $\forall \sigma_k \in \Sigma_k (k = 1, 2, ..., n)$, $\pi_k(\sigma_k^*, \sigma_{-k}^*) \geq \pi_k(\sigma_k, \sigma_{-k}^*)$.

Based on the Nash equilibrium point definition and payoff functions, we can derive the theorem below.

Theorem 1. In a witness game, the only two Nash equilibrium points are following strategy profiles:

- $\sigma^* = (\sigma_1^*, \sigma_2^*, ..., \sigma_n^*)$, of which $\forall w_k \in SW$, $\sigma_k^* = \sigma_k^{(r)}$;
- $\sigma^* = (\sigma_1^*, \sigma_2^*, ..., \sigma_n^*)$, of which $\forall w_k \in SW$, $\sigma_k^* = \sigma_k^{(s)}$.

Proof. According to Definition 1 and 2, in a N-witness game, $N \geq 3$, $N/2 < M \leq N - 1$ and $M \geq 2$, where $N, M \in \mathbb{N}$.

For the strategy profile of $\forall w_k \in SW$, $\sigma_k^* = \sigma_k^{(r)}$, which means $\|W_{report}\| = N > M$. The SLA violation status is therefore violated, $SLA_{status} = V$. According to payoff functions design in Definition 3, for $\forall w_k$, its revenue is $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = 10$. If any w_k chooses the other action, *S*ilence, instead of *R*eport. The final status of SLA, however, would not be modified, due to $\|W_{report}\| = N-1 \geq M$. Then, w_k's revenue is $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 0 < 10 = \pi_k(\sigma_k^{(r)}, \sigma_{-k}^*)$. According to Definition 5, this strategy profile is a Nash equilibrium point.

Analogously, for the strategy profile of $\forall w_k \in SW$, $\sigma_k^* = \sigma_k^{(s)}$, which means $\|W_{report}\| = 0 < 2 \leq M$. The SLA violation status is therefore not violated, $SLA_{status} = C$. According to payoff functions design in Definition 3, for $\forall w_k$, its revenue is $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 1$. If any w_k chooses the other action, *R*eport, instead of *S*ilence. The final status of SLA, however, would not be modified, due to $\|W_{report}\| = 1 < 2 \leq M$. Then, w_k's revenue is $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = -1 < 1 = \pi_k(\sigma_k^{(s)}, \sigma_{-k}^*)$. According to Definition 5, this strategy profile is also a Nash equilibrium point.

For all the other strategy profiles, they are all a mix of actions, *R*eport and *S*ilence. It means $W_{report} \neq \emptyset$ and $W_{silence} \neq \emptyset$. When $SLA_{status} = V$, i.e., $\|W_{report}\| \geq M$, there always $\exists w_k \in W_{silence}$, it can change the action to *R*eport. But the SLA status would not change, because of $\|W_{report}\| + 1 > M$. Hence, w_k increases its revenue, from $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 0$ to $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = 10$. On the other hand, when $SLA_{status} = C$, i.e., $\|W_{report}\| < M$, there always $\exists w_k \in W_{report}$, it can change the action to *S*ilence. But the SLA status would not change, because of $\|W_{report}\| - 1 < M$. Hence, w_k increases its revenue, from $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = -1$ to $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 1$. These counterexamples demonstrate all these strategy profiles are not Nash equilibrium points.

Therefore, in a witness game, there are two and only two Nash equilibrium points. They are $\sigma^* = (\sigma_1^{(r)}, \sigma_2^{(r)}, ..., \sigma_n^{(r)})$ and $\sigma^* = (\sigma_1^{(s)}, \sigma_2^{(s)}, ..., \sigma_n^{(s)})$. □

We take the basic $three$-witness game as an example, where $N = 3$. Therefore, M can only equal to 2 based on Definition 2. Table I shows payoff functions according to our previous definitions. The value element in Table I is the vector of corresponding payoff function values. It is represented as (π_1, π_2, π_3). According to Theorem 1, Nash equilibrium points in this game are (10, 10, 10) and (1, 1, 1) respectively.

Based on above analysis, for a rational and selfish witness, who wants to maximize its revenue through offering services, would have to behave as follows in this game. If there is a violation happening, the witness knows that most of other witnesses are more likely to report this event to gain more revenue. Hence, the higher revenue pushes the witness to report this event. On the contrary, if there is no violation, the witness knows that most of other witnesses are more likely to keep silence. Although the witness wants to achieve the highest revenue, it has to take a great risk to pay a penalty for its fraudulent behavior. From the global view, when there is no violation, all the witnesses prefer to keep silence in order to stay at the Nash equilibrium point, $(\sigma_1^{(s)}, \sigma_2^{(s)}, ..., \sigma_n^{(s)})$. Then the violation acts as a signal to push them achieving another Nash equilibrium point, $(\sigma_1^{(r)}, \sigma_2^{(r)}, ..., \sigma_n^{(r)})$, with much higher revenue. At the same time, they tell the truth about the service violation.

Therefore, it is not the witness who wants to tell the truth. Instead, it has to be honest, in order to maximize its revenue.

C. Witness Audit

The sortition algorithm, Algorithm 1, makes sure that the selected witnesses are independent to a great extent. The payoff function design stimulates the witness to tell the truth. However, an auditing mechanism is still needed to ensure that the malicious or irrational witness can be detected and kicked out from the witness pool. As all the interactions with the smart contract, i.e., transactions, are public and permanently stored on the blockchain, it is possible to audit a witness through its behavior history, instead of estimating reputation from others' feedback [20]. We analyze that there are two types of malicious witnesses: lazy witness and sacrificed witness.

Lazy witness refers to the one, who prefers not to report the violation. Because there is a case that the higher incentive for reporting violation is not enough to motivate the lazy one. They can choose the strategy not to really monitor the service.

TABLE I: Payoff Functions for a $three$-witness Game

w_1	w_3			
	$\sigma_3^{(r)}$: *R*eport		$\sigma_3^{(s)}$: *S*ilence	
	w_2		w_2	
	$\sigma_2^{(r)}$: *R*eport	$\sigma_2^{(s)}$: *S*ilence	$\sigma_2^{(r)}$: *R*eport	$\sigma_2^{(s)}$: *S*ilence
$\sigma_1^{(r)}$: *R*eport	(10, 10, 10)	(10, 0, 10)	(10, 10, 0)	(-1, 1, 1)
$\sigma_1^{(s)}$: *S*ilence	(0, 10, 10)	(1, 1, -1)	(1, -1, 1)	(1, 1, 1)

Then they always keep silence and never report the violation. With this strategy, they would not pay the penalty, if the final status of the SLA is violated. And considering violation is not a normal event, the lazy witness is still able to gain some revenue via multiple "games".

Sacrificed witness refers to the one, who always reports at a specific time stamp. For instance, w_k always reports the violation within one minute after the SLA starts. Though the witness may pay a lot of penalty for its malicious behavior in the beginning, it can show other witnesses its behavior pattern from its history later on. In some sense, it is able to imply others that it would report at some time stamp. Then, it can most likely gain the maximum revenue, as long as others have analyzed its behavior pattern.

To audit these fraudulent witnesses, we combine the reputation value of the witness. For example, when the violation is confirmed, the reputation of the witness, who does not report, would decrease by 1. It is the same with the one, who reports the violation but not finally confirmed. When the reputation value of the witness decreases to zero, that witness would also be blocked automatically by the sortition algorithm. The lazy witness and sacrificed witness, therefore, are blocked soon. It is also worth mentioning that these auditing mechanisms are also implemented in the smart contract, in order to avoid a third party to dominate the judgement.

V. Prototype Implementation and Experiments

According to the witness model and the payoff function design, we implement a prototype system based on the smart contracts of Ethereum. We leverage the programming language, Solidity[6], provided by Ethereum, to program smart contracts. Overall, there are three roles and two types of smart contracts in our SLA enforcement system. Roles include the traditional Provider and Customer, as well as the introduced Witness. The smart contracts include the witness-pool smart contract and the SLA smart contract. In this section, we firstly illustrate the state transition in that two types of smart contracts, respectively. Via this, we describe the detailed functionalities of the interfaces in the smart contracts and show how they are leveraged to transit the states. Afterwards, we show some experimental studies on the transaction cost of these interfaces on the Ethereum test net, "Rinkeby".

Interfaces designed in both two smart contracts are named as the text on the arrow in Figure 2 and 3. The format of the text is '$R_{role} \rightarrow [C_{type}::]N_{interface}$'. It means that only the role R_{role} can invoke the interface, $N_{interface}$, which is defined in the smart contract of C_{type}. This is achieved by the checking mechanism, which is the property of the programming language provided by Ethereum. Therefore, the smart contract restricts that only the specified role is able to interact with the smart contract in certain state. The representation of the R_{role} are W for Witness, P for Provider, C for Customer, and SC for a generated SLA Smart Contract. To express C_{type}, WP is for the witness-pool type of smart contract and SLA is for the generated smart contract to enforce a specific SLA.

[6]http://solidity.readthedocs.io

A. Witness-pool Smart Contract Implementation

In this part, we focus on implementation details about the witness-pool smart contract, especially the witness management. Figure 2 illustrates states of a witness role defined in the smart contract. It includes four states: "Online", "Offline", "Candidate" and "Busy". The state transition of witness is as follows.

After registration in the witness pool, only the witness itself can turn its state into "Online". It is then probably selected by certain SLA smart contract. Hence, it needs to monitor its own state on the blockchain continuously. This is feasible, as the read-only operation does not need any transaction fee. Once it is selected by certain smart contract through performing sortition algorithm 1, its state turns into "Candidate". Within a confirmation time window, e.g., 2 minutes, the witness candidate can look through the SLA smart contract, which makes the selection, and decides whether to confirm or reject this selection. If it rejects, the provider of the SLA smart contract has to perform another sortition. Otherwise, its state turns into "Busy", after it invokes the interface, "witnessConfirm", of the SLA smart contract. By the end of each SLA lifecycle iteration, the witness has the right to actively leave the SLA contract by leveraging the interface "witnessRelease". On the other hand, the witness can also be passively released from the SLA contract. Because the provider invokes the interface "resetSLA" to dismiss the witness committee. Finally, the witness can "turnOff" to avoid being selected when it is not available to the Internet.

In order to prevent some malicious intentions, we bring in a reputation value for each witness to measure their behaviors. Firstly, each witness has an initial reputation value of R_{init} at registration, which could be a predetermined constant or a value depending on its stake or some deposit. However, for instance, some witness may not turn its state into "Offline", when it is not actually available or does not frequently check its state. Then, when the witness is chosen by some SLA smart contract, it would not be able to confirm the selection and join within the confirmation time window. In this case, the witness would not be chosen again, since its state becomes "Candidate". To reverse back to the state of "Online", in which

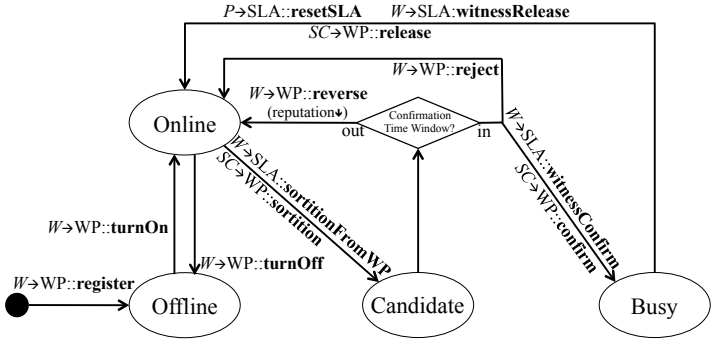

Fig. 2: State Transition Diagram of a Witness in the Witness-pool Smart Contract

it can be selected, the witness has to leverage the interface, "reverse". However, its reputation value would decrease by 10 because of its above behavior. If this value becomes zero or less, it would be permanently blocked by the sortition process according to Algorithm 1.

B. SLA Smart Contract Implementation

Figure 3 shows the SLA state transition to implement a specific SLA smart contract enforcement. This type of smart contract is generated by the witness-pool smart contract. All the interfaces annotated in this figure belongs to this type of SLA smart contract. Hence, we omit the definition scope 'C_{type}::'. There are five states: "Fresh", "Init", "Active", "Violated" and "Completed", shown as circles in Figure 3. The dash arrows demonstrate the state transition path when violation happens. The three squares in the figure represent the corresponding roles in this smart contract. In the end of SLA, they can withdraw the revenue respectively.

It is also worth mentioning that the smart contract on blockchain cannot run itself. The state transition must be triggered by some interfaces and it takes some cost to execute. Therefore, we design the interface for the role, who is the greatest beneficiary in some cases, to modify the state. Because they have the motivation to perform the state transition. For example, when the service duration ends normally, the provider is the greatest beneficiary to gain the entire service fee. It must actively leverage the interface, "providerEndNSLAand-Withdraw", to end the normal SLA and withdraw its own revenue. Meanwhile, it divides the prepaid money as the payoff function design in Section IV-B to different witnesses. Afterwards, other roles are able to withdraw their parts of revenue. Analogously, when there is a violation, the customer is the most motivated one to gain the compensation fee. It can leverage, "customerEndVSLAandWithdraw", to end the violated SLA and transit the state from "Violated" to "Completed".

C. Experimental Study

In order to test all the functionalities of our model and system design, we deploy the implemented smart contracts

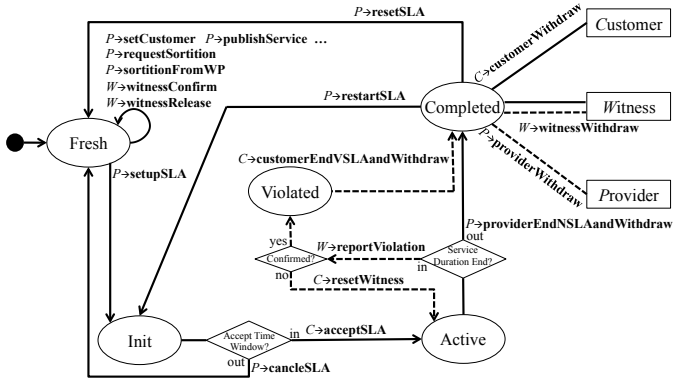

Fig. 3: State Transition Diagram of SLA Lifecycle for a Specific SLA Smart Contract

on the test net of Ethereum blockchain, "Rinkeby". It is a world-wide blockchain test net for developers to debug the smart contract. The 'Ether', which is the cryptocurrency of Ethereum, does not worth real value on the test net and can be applied for debugging. Hence, we generate several accounts on "Rinkeby" to simulate different roles, i.e., the provider, customer and witnesses. We leverage the retrieved 'Ether' on each simulated account to execute the interfaces and prepay different types of fees according to the model. To conduct the experiment, we first deploy the basic witness-pool smart contract, and make all the accounts register to the witness pool. The provider then generates a SLA smart contract to start the SLA lifecycle with the customer. Afterwards, we test all possible scenarios to exploit and validate the functionality of different interfaces. The results demonstrate our system implementation satisfies our model and payoff function design.

The trust part of the system is proved by game theory and ensured by the unbiased sortition algorithm, whose credibility is endorsed by the blockchain technique. Therefore, we mainly analyze some performance information from our experimental study. Thereinto, the performance refers to the complexity of each interface in the smart contract. It determines the transaction fee needed to pay the miner in Ethereum. Because the miner needs to execute the program defined in the interface for validation and consumes the electricity power. The more complex of the interface is, the more transaction fee required when it is invoked. This is measured as 'Gas' defined in Ethereum, which is a unit to refer how much work taken by the miner when executing the transaction. The transaction fee is the product of gas consumption and the gas price for each unit. Hence, the gas consumption is similar no matter on test net or main net. We therefore record all the gas consumption for each interface from the transaction history of the experiment.

Figure 4 illustrates results of the experimental study. It can be derived that, compared with the customer and the witness, the provider tends to require more gas in the entire SLA lifecycle. The interfaces of customer and witness consume less. This fits our model design and reality. Because in most cases, the provider earns the most revenue through offering service. It has the incentive to proceed the lifecycle. The light weight gas consumption for witness role is also able to convince blockchain users to take part in the system to work as a witness. Moreover, these gas consumption values are achieved through experiments based on current implementation. There is still possible space to further optimize the interface implementation to lower the gas consumption.

VI. Conclusion

In this paper, a witness model is proposed for cloud SLA enforcement and the payoff function is specially designed for each witness. We leverage the game theory to analysis that the witness has to offer honest monitoring service in order to maximize its own revenue. Finally, a prototype system is fully implemented using smart contracts of Ethereum to realize the witness model. Not only the SLA enforcement lifecycle but also the witness management of a witness pool

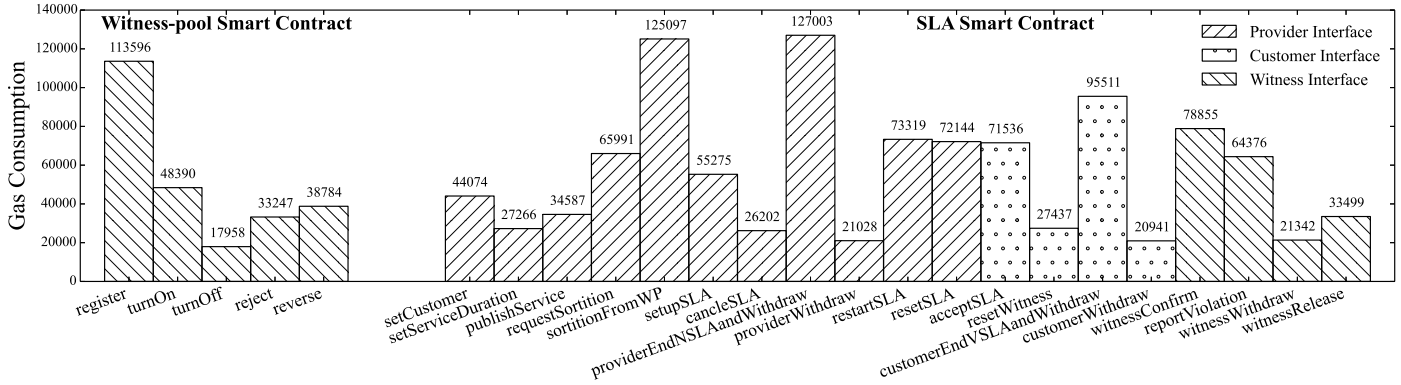

Fig. 4: The Gas Consumption of Each Interface in Smart Contracts

are implemented with the smart contract. The experimental study demonstrates the feasibility of our model and shows the system performance. Via this way, the trust problem is transferred into economic issues. It is not the witness itself would like to be honest, but the economic principles force them to tell the truth. To the best of our knowledge, this is also the first work that applies economic principles to achieving trustworthy consensus among "oracles", who carry the event data onto the blockchain.

For the future work, there are mainly two directions: on-chain and off-chain. For the on-chain part of work, we are going to further optimize the interface implementation to reduce the gas consumption and enrich the functionalities of our smart contracts. In addition, some more scenarios should be considered to apply our model. For the off-chain part of work, user-friendly tools are going to be developed for each role in the system to monitor the state on the chain and perform their corresponding interactions. On the other hand, it can also be combined with our cloud application DevOps framework, CloudsStorm[7] [21], to construct the witness ecosystem. The vision is to insure the cloud performance and reduce the risks for applications through automated SLA.

ACKNOWLEDGMENT

This research is funded by the EU Horizon 2020 research and innovation program under grant agreements 825134 (ARTICONF project), 654182 (ENVRIPLUS project) and 824068 (ENVRI-FAIR project).

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] Z. Zhao, A. Taal, A. Jones, I. Taylor, V. Stankovski, I. G. Vega, F. J. Hidalgo, G. Suciu, A. Ulisses, P. Ferreira *et al.*, "A software workbench for interactive, time critical and highly self-adaptive cloud applications (switch)," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 1181–1184.

[3] F. Faniyi and R. Bahsoon, "A systematic review of service level management in the cloud," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 43, 2016.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[6] H. Nakashima and M. Aoyama, "An automation method of sla contract of web apis and its platform based on blockchain concept," in *Cognitive Computing (ICCC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 32–39.

[7] (2014) Ethereum and Oracles. [Online]. Available: https://blog.ethereum.org/2014/07/22/ethereum-and-oracles/

[8] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, "Renegotiation in service level agreement management for a cloud-based system," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 51, 2015.

[9] N. Ghosh and S. K. Ghosh, "An approach to identify and monitor sla parameters for storage-as-a-service cloud delivery model," in *Globecom Workshops (GC Wkshps), 2012 IEEE*. IEEE, 2012, pp. 724–729.

[10] C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortes, and M. Rodriguez, "Comprehensive explanation of sla violations at runtime," *IEEE Transactions on Services Computing*, vol. 7, no. 2, pp. 168–183, 2014.

[11] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart contract templates: foundations, design landscape and research directions," *arXiv preprint arXiv:1608.00771*, 2016.

[12] V. Scoca, R. B. Uriarte, and R. De Nicola, "Smart contract negotiation in cloud computing," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 2017, pp. 592–599.

[13] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*. ACM, 2016, pp. 270–282.

[14] H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, and S. Čapkun, "Tls-n: Non-repudiation over tls enabling ubiquitous content signing for disintermediation," *IACR ePrint report*, vol. 578, 2017.

[15] S. Ellis, A. Juels, and S. Nazarov, "Chainlink: A decentralized oracle network," *white paper*, 2017.

[16] H. Zhou, C. de Laat, and Z. Zhao, "Trustworthy cloud service level agreement enforcement with blockchain based smart contract," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 255–260.

[17] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," *arXiv preprint arXiv:1801.03998*, 2018.

[18] K. Binmore, *Game theory: a very short introduction*. Oxford University Press, 2007, vol. 173.

[19] J. F. Nash *et al.*, "Equilibrium points in n-person games," *Proceedings of the national academy of sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[20] X. Wang, J. Su, X. Hu, C. Wu, and H. Zhou, "Trust model for cloud systems with self variance evaluation," in *Security, Privacy and Trust in Cloud Systems*. Springer, 2014, pp. 283–309.

[21] H. Zhou, Y. Hu, J. Su, C. de Laat, and Z. Zhao, "Cloudsstorm: An application-driven framework to enhance the programmability and controllability of cloud virtual infrastructures," in *International Conference on Cloud Computing*. Springer, 2018, pp. 265–280.

[7]https://cloudsstorm.github.io/