



UvA-DARE (Digital Academic Repository)

Distributed service-level agreement management with smart contracts and blockchain

Uriarte, R.B.; Zhou, H.; Kritikos, K.; Shi, Z.; Zhao, Z.; De Nicola, R.

DOI

[10.1002/cpe.5800](https://doi.org/10.1002/cpe.5800)

Publication date

2021

Document Version

Final published version

Published in

Concurrency and Computation: Practice and Experience

License

CC BY

[Link to publication](#)

Citation for published version (APA):

Uriarte, R. B., Zhou, H., Kritikos, K., Shi, Z., Zhao, Z., & De Nicola, R. (2021). Distributed service-level agreement management with smart contracts and blockchain. *Concurrency and Computation: Practice and Experience*, 33(14), [e5800]. <https://doi.org/10.1002/cpe.5800>

General rights




It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (<https://dare.uva.nl>)

Distributed service-level agreement management with smart contracts and blockchain

Rafael Brundo Uriarte^{1,2}  | Huan Zhou^{3,4}  | Kyriakos Kritikos⁵ | Zeshun Shi⁴ | Zhiming Zhao⁴  | Rocco De Nicola^{1,6}

¹IMT School for Advanced Studies Lucca, Lucca, Italy

²Vienna University of Technology, Vienna, Austria

³School of Computer Science, National University of Defense Technology, Changsha, China

⁴Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands

⁵Information Systems Laboratory, ICS-FORTH, Heraklion, Greece

⁶Cybersecurity National Laboratory, CINI, Roma, Italy

Correspondence

Rafael Brundo Uriarte, IMT School for Advanced Studies Lucca, Lucca, Italy
Zhiming Zhao, Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
Emails: rafael.uriarte@gmail.com and z.zhao@uva.nl

Funding information

BLUECLOUD project, Grant/Award Number: 862409; ENVIR-FAIR project, Grant/Award Number: 824068; H2020 Future and Emerging Technologies, Grant/Award Number: 825134; H2020 Marie Skłodowska-Curie Actions, Grant/Award Number: 838949; LifeWatch ERIC and by MIUR project PRIN 2017FTXR7S IT-MATTERS (Methods and Tools for Trustworthy Smart Systems); ARTICONF project, Grant/Award Number: 825134

Summary

The current cloud market is dominated by a few providers, which offer cloud services in a take-it-or-leave-it manner. However, the dynamism and uncertainty of cloud environments may require the change over time of both application requirements and service capabilities. The current service-level agreement (SLA) management solutions cannot easily guarantee a trustworthy, distributed SLA adaptation due to the centralized authority of the cloud provider who could also misbehave to pursue individual goals. To address the above issues, we propose a novel SLA management framework, which facilitates the specification and enforcement of dynamic SLAs that enable one to describe how, and under which conditions, the offered service level can change over time. The proposed framework relies on a two-level blockchain architecture. At the first level, the smart SLA is transformed into a smart contract that dynamically guides service provisioning. At the second level, a permissioned blockchain is built through a federation of monitoring entities to generate objective measurements for the smart SLA/contract assessment. The scalability of this permissioned blockchain is also thoroughly evaluated. The proposed framework enables creating open distributed clouds, which offer manageable and dynamic services, and facilitates cost reduction for cloud consumers, while it increases flexibility in resource management and trust in the offered cloud services.

KEYWORDS

blockchain, cloud computing, distributed ledgers, distributed management, service-level agreements, smart contracts

1 | INTRODUCTION

Despite the potential cloud computing's benefits in terms of providing on-demand computing services, some limitations and barriers prevent its wide adoption. One of the most important limitations is related to quality of service (QoS) guarantees, that, if offered at all, are defined in natural language, are concerned mostly with availability, and are specified just once and for all. However, cloud services are offered in dynamic environments and have their service level constantly changing over time, while the delivered service level might not be objectively measured. This often leads to disputes and moves the monitoring burden on consumers. At the same time, consumers and providers cannot easily change the terms of the contract

Abbreviations: IaaS, infrastructure-as-a-service; SLA, service-level agreements; VM, virtual machine. All authors have contributed equally to this work.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2020 The Authors. *Concurrency and Computation: Practice and Experience* published by John Wiley & Sons Ltd.

when they have an increase in workload, faults, incidents, or change of requirements. For example, currently contracts using the IaaS cloud model just define that a specific instance of a VM type will be made available to the user. This means that new instances will have to be actually associated with new contracts, leading to an increase in the amount of work needed for contract management. Even when contracts enable users to create multiple instances up to a certain limit for each VM type, cloud bursting and vertical scaling scenarios, which require using a different number of instances of different VM types, cannot be properly considered. Currently, to handle this situation, it would be necessary to terminate the active contract and to create a new one. The same applies to non-hardware-related SLA terms, for example, response time, availability, monitoring frequency, penalties and security, and to other cloud service models (eg, software-as-a-service [SaaS] and platform-as-a-service [PaaS]).

The lack of flexibility of contracts and objective service measurement also increases the lock-in effect.¹ This means that in the presence of uncertainty about the moving costs, users are reluctant to move to another cloud provider even if their current provider does not supply its service(s) at the promised (quality) level. Furthermore, the reluctance in migrating is reinforced by the extra effort, including the adoption of different cloud-specific technologies, data migration, and the additional management overhead needed to stop old contracts and start new ones. For a complete analysis of the benefits of dynamic service-level agreements (SLAs) and several use case scenarios, we refer the interested reader to Reference 2.

Smart contracts and distributed ledgers (eg, blockchain) have the potential to address the above issues. Smart contracts are agreements described in an executable language that enables trusted transactions. One of their specific features is that they can be consistently executed by a network of mutually distrusted nodes, without the arbitration of a trusted authority.³ Smart contracts have great expressive power and are used in different areas, from finance to cloud services. Blockchain⁴ is an implementation of the concept of distributed ledgers that securely registers transactions, even for completely decentralized systems, and removes the need for trusted intermediaries, such that the only trusted element is the distributed and verifiable computing system.

Using a combination of smart contracts and blockchain in the cloud setting can lower costs and bring predictable results.^{5,6} This combination offers a purely decentralized system that handles cloud services and applications along their whole life cycle and does not depend on the trustworthiness of individual partners. Thus, the combination of smart contracts and blockchain paves the way to a truly open cloud market, which has been hindered by the issues of security and trust. Furthermore, this integration makes it possible to verify that the delivered service is the desired one and that the agreed QoS is enforced.

Blockchain and smart contracts have been used in the cloud domain only recently and with a focus on commercial targets.⁶ In fact, the design choices of the most advanced projects in the area have been taken on an ad hoc basis with no systematic study, mainly concentrating on delivering a platform without any QoS guarantees.⁶ To address the above issues, we propose an SLA management framework for smart contracts, which relies on a two-level architecture.

Our two-level architecture is built using both *permissionless* and *permissioned* blockchains, with the former allowing anyone to participate and the latter restricting such a participation to selected users. On the first level of our architecture, the dynamic, smart SLA agreed by two signatory parties is transformed into a smart contract, which is executed in a permissionless blockchain to handle the indisputable management of the respective cloud service. The second level of the architecture supports the first one through a permissioned blockchain, which is instrumental in raising the trust in the SLA assessment and in guaranteeing that a smart contract moves to a new state only when the agreed conditions hold. The permissioned blockchain is formed by a randomly selected set of oracles, which perform service monitoring and provide objective measurements and undisputed transactions. The outcome of these transactions is compared with the smart contracts of the first level to determine violations of service-level objectives (SLOs) or the need to change the contract state.

The proposed framework relies on our previous work^{2,7} that (i) introduced a new SLA definition language called SLAC covering the distinctive aspects of cloud services and (ii) offered novel mechanisms for dealing with the SLA dynamism. These mechanisms offer the possibility of properly formalizing the elasticity of cloud services while guaranteeing flexibility to the involved parties. However, our previous work relied on service management frameworks based on the central authority of the cloud service provider, which could misbehave to, for example, increase its profit. The framework we propose here, instead, follows a model-driven architecture to transform dynamic SLAs into smart contracts and automate the distributed SLA enforcement. It offers a neutral and deterministic intermediary that manages the collection and verification of monitoring data and the changes in the services levels and sets up services billing and payment. Thus, services are measured objectively while dynamic management are automated.

The first level of the proposed architecture has been presented in Reference 8; the new framework provides an extension, which offers the possibility of objectively assessing smart SLAs/contracts. Another contribution of this article is a thorough discussion of the challenges in implementing the proposed architecture and a description of the new implementation of both architectural levels. A final contribution of this article is the thorough evaluation of the second architectural level with a focus on scalability. Such an evaluation has been conducted by using specific workloads on different clouds and data centers and by varying the communication bandwidth between VM's. The main findings of the evaluation are the followings: (i) the stability of the performance of a permissioned blockchain depends on the network and on resources utilization in data centers, (ii) the use of better VMs significantly increases performance but slightly decreases stability, (iii) the throughput of many management scenarios can be controlled by regulating the network bandwidth and by carefully selecting VMs.

The advantages of our SLA management framework based on smart contracts and on the two-level blockchain architecture are manifold. In particular, it:

- (a) supports environments where parties do not need to trust each other; thus lowering market barriers;
- (b) automates the SLA management life cycle;
- (c) brings transparency to service provisioning by clearly defining all rules for SLA management in public smart contracts;
- (d) enables the objective measurement of cloud services;
- (e) introduces flexibility in the cloud service/resource management, thus enabling cloud providers to change the delivered service levels on demand;
- (f) adds flexibility for consumers who can require service-level changes;
- (g) reduces the probability of service renegotiation, thanks to the dynamic service-level adjustment.

The rest of this article is organized as follows. Section 2 reviews some related work and provides some essential background knowledge. Section 3 presents the overall architecture of the envisioned framework and its first architectural level. The second architectural level is presented in Section 4. The experiments and the evaluation results are presented in Section 5. The final section summarizes the contents of this article and suggests possible directions for future work.

2 | DECENTRALIZED MULTICLOUD ENVIRONMENTS: BACKGROUND AND RELATED WORK

A blockchain records digital events or transactions in a distributed database shared among the network participants. These transactions are nonerasable; cryptographic proofs are used to attest authorship, and they are verified by a consensus protocol, which is usually based on the majority of the participants in the system. Each transaction is broadcasted to every node in the network and, after verification,⁹ it is recorded in a public ledger. This paradigm has the potential to automate and replace third-party entities that guarantee security and privacy in digital transactions.

Smart contracts, instead, are computer programs that automatically execute the terms of a contract. When a condition is met, an action is automatically executed in a transparent and auditable manner, for example, the payment for a service. They can be used in different contexts, such as financial, notary, and games (betting and game industry).¹⁰

Successful implementations of such technologies, for example, Bitcoin and Ethereum, have encouraged the development of projects focusing on different types of decentralized solutions. In the cloud context, the situation is similar, and several decentralized solutions for the cloud/fog market are emerging. The most advanced blockchain-based cloud projects are Golem*, iExec†, and SONM‡. iExec is the only project that aims at covering QoS guarantees, but it relies on standard SLAs and does not cover dynamic aspects. Moreover, considering the public roadmap of these projects, there are no plans to support SLAs within smart contracts.⁶ On the contrary, according to their adopted architecture, SLAs would probably be defined and controlled by the provider without smart contracts. In our previous work,⁶ we have presented a thorough analysis of these projects, discussed the challenges involved in building decentralized cloud/fog solutions and supplied an overview of the standards applicable in the area.

Figure 1 depicts a high-level architecture for blockchain-based clouds, which comprises two main layers: (i) the transaction network (also referred as the main chain) and (ii) the side-chain network.

The transaction network uses the blockchain to improve trust and security in the decentralized system to safely register transactions and regulate payments. In the case of decentralized cloud solutions, smart contracts are established and enforced, the transactions are registered and managed, and the reputation of the participants is maintained. All operations in the blockchain must be deterministic, that is, the same operation performed at different nodes of the network must return the same result(s); otherwise, inconsistency in the blockchain would be introduced, which can lead to failure in results validation and storage.

Since transaction networks have several limitations related to high cost of storing and processing data, reduced scalability, and low transaction confirmation speed, alternatives, such as off-chain or side-chains networks, may be adopted.¹¹ The term *off-chain* refers to computations executed outside the blockchain. The major drawback of off-chain computations is that either users of the transaction network (eg, the cloud consumer) must trust the parties running the computation or specific methodologies have to be devised to verify the results (like the one proposed in this article). Moreover, instead of off-chain execution, separated blockchain-based networks, called *side-chains*, can be created with restricted access (permissioned blockchain) but such that their transactions and smart contracts can be publicly verified by their users.

*<https://golem.network/>

†<http://iex.ec/>

‡<https://sonm.com/>

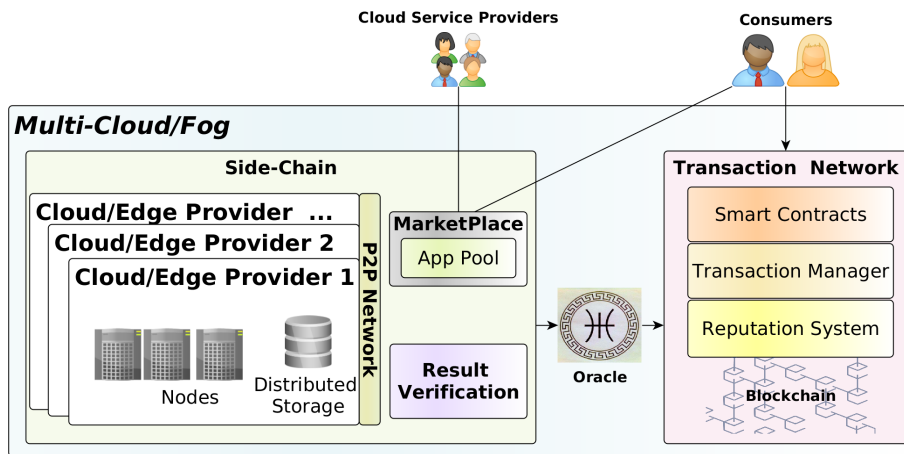


FIGURE 1 Multicloud architecture

In this article, we use both types of alternative chains: side-chains to verify the results of computations and reach consensus, and off-chain execution for computation-intensive tasks that require external data. The latter are executed using resources from third-party providers, who are rewarded for their work. More specifically, the following tasks are executed off-chain: (i) the actual services, (ii) the marketplace module, which supports software verification and negotiation, (iii) the service monitoring, and (iv) the verification module, which checks whether the service is executed according to the specification and whether its outcome is correct.

In the high-level architecture of Figure 1, *cloud service providers* create and make their cloud applications/services available in the *marketplace*. Then, to access such services, *consumers* negotiate the terms of the respective SLA and, as a result, smart contracts that regulate the provisioning of the services are generated. However, off-chain operations could be nondeterministic. This different behavior of the blockchain used in the transaction network and the side or off-chain requires an interface, the so-called *oracle*, which, after retrieving the relevant information, evaluates it, by running a consensus protocol, and transmits the information to the transaction network. Specifically, the oracle receives (by executing service monitors drawn from the service contract) service monitoring data and inserts them into the transaction network, which uses them for smart contract execution.

An oracle implementation could rely on retrieving data directly from trusted services, such as Provable⁴, Town Crier¹² and TLS-N.¹³ These, however, are centralized services that suffer from the well-known single point of failure (SPoF) problem and deviate from the discipline of decentralization, which is widely adopted in the blockchain. ChainLink¹⁴ works on distributed oracles that carry the data onto the chain or can trigger transactions only when an agreement is achieved among all the oracles. However, neither ChainLink nor any of the previously mentioned solutions support any kind of QoS monitoring.

In traditional clouds, SLAs specify the service level to be guaranteed by a cloud service. In this respect, many SLA specification languages¹⁵ and SLA automation frameworks have been proposed.¹⁶⁻¹⁹ These languages allow the definition of a single and nonmodifiable service level.¹⁵ In our previous work,^{2,20} we addressed this problem by introducing a new mechanism to specify multiple service levels as well as the conditions enabling to move from one service level to another. This mechanism opens up the opportunity to use SLAs in blockchain environments and for further extensions. To the best of our knowledge, only²¹ considers smart contract-based SLA management. However, the authors focus on the specification of APIs, while offering only a high-level description of the blockchain-based SLA management. Many important components and methods, such as consensus and validation techniques, mechanisms for data collection, payment and billing processes, are not even mentioned.

We have also proposed a smart contract transformation module, named SLA2SC,⁸ that acts as a neutral third party and mediates service provision. This third-party component is similar to a third-party cloud service, which is registered in the transaction network but is not owned/controlled by any provider. In fact, it is an immutable and auditable algorithm that is available in the blockchain and comprises methods, which are accessible by everyone in the transaction network.

3 | FROM SLAS TO SMART CONTRACTS

The definition of quality of service (QoS) is essential for consumers, especially those who require guarantees for outsourcing their needed resources or for executing important business processes and applications. SLAs formalize the terms for the service's provisioning and its quality. In this context, with SLAC,² the involved parties can define and cover the particularities of the cloud domain, including the capacity to model the evolution of service requirements/capabilities over time. Although SLAC virtually covers all needs directly related to the cloud service

⁴<https://provable.xyz/>

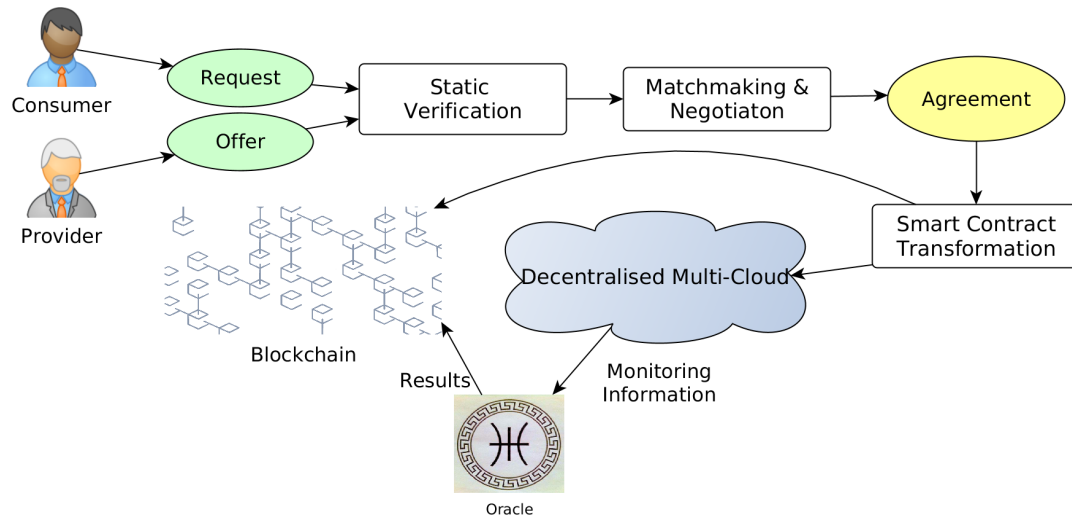


FIGURE 2 From service definition to deployment

provisioning domain, its related theoretical and practical frameworks for the evaluation of SLA agreements have been designed as centralized and thus rely on the trustworthiness of the service provider. Lowering the entry barriers and supporting decentralized open markets are still big challenges.

In this article, we propose a framework, named cloud service smart contract manager (CSSCM), for the definition and enforcement of SLAs by transforming them into smart contracts, which are then executed in the blockchain. The process is mainly divided into the off-chain and blockchain (transaction network) phases. Data processing in public blockchains is costly and collecting data from external sources requires trusted parties or a consensus protocol. For example, the dynamic needs of the parties and the possibly high number of providers imply that match-making and negotiation must be based on up-to-date data about all cloud offers; thus, a very large number of (contract) states needs to be considered. For these reasons, we opted for performing off-chain negotiation and static verification of SLAs. Only after the negotiation phase, and thus after the SLA is defined, verified and agreed, a smart contract is derived from it, which is then executed, enforced, and billed in the blockchain.

Figure 2 illustrates the major steps of the aforementioned process. The involved parties define offers and requests in the SLAC language, then their consistency is verified via static analysis. Subsequently, a *Broker* or the *Marketplace* matches and negotiates the request and its related offers by using the SLAC framework to produce a final SLA. Such an SLA is then transformed into a smart contract and executed in the blockchain. Finally, multiple oracles monitor the service execution and achieve a consensus on the actual measurements derived by a predefined consensus protocol. Once consensus is reached, valid monitoring data are added to the blockchain to enable the evaluation of the conditions of the smart contract. Our consensus protocol for QoS verification in the cloud is presented and discussed in the next section.

Figure 3 shows the flow chart of cloud service management by considering the actions performed by the software components and by the involved actors. First, the involved parties (providers and consumers) create an account in the system, then the consumer sends the agreed SLA and the provider confirms it. Once confirmed, SLA2SC parses the SLA and creates a new smart contract in the transaction network, that is, the service smart contract, which autonomously manages the service. This service smart contract starts its execution by requesting the provider to deploy the corresponding resources. After that, service results and QoS data are collected by oracles (auditors), which aim at consensus to assess their correctness. Finally, the verified QoS data are sent to the service smart contract for evaluation.

The definition of correctness and the value of the result to be submitted to the smart contract depend on the consensus protocol and on its implementation. An important challenge to be faced when using smart contracts for QoS verification is the setting up of a decentralized mechanism for collecting trustworthy monitoring data. Entities called oracles have been used to assess the quality of gathered monitoring data and feed them into the consensus mechanism of a blockchain.²² To this end, we have introduced the second level in our SLA management architecture, which is able to utilize smart contracts on a permissioned blockchain to achieve trustworthy cloud monitoring and validation.

Currently, we take the perspective that the involved oracles only verify whether SLO violations have occurred by considering the majority of votes. In this respect, the service smart contract just takes the necessary actions for each verified SLO received, such as changes in the service deployed (eg, scaling up and down) and contract termination due to potential violations or misbehavior. Furthermore, the service smart contract can receive requests of changes from the signatory parties, which, if accepted, will be subsequently applied (and reflected on the contract state).

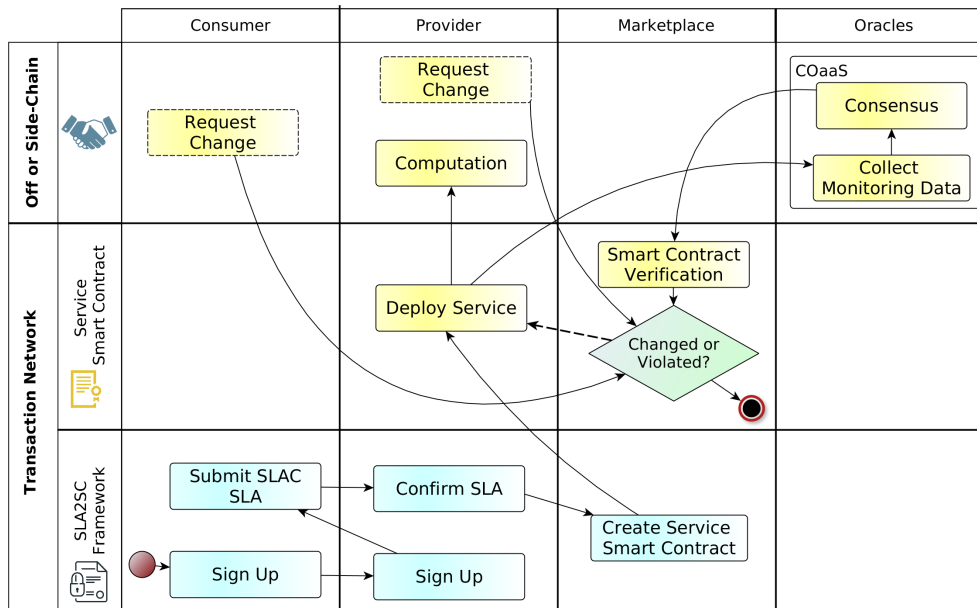


FIGURE 3 CSSCM framework

3.1 | CSSCM framework implementation

Here, we briefly overview the contract creation and the dynamic parts of the implementation of a prototype for supporting SLA management in the blockchain using the contract-oriented programming language Solidity⁵ and the Ethereum platform.

The main tasks of the CSSCM framework are (i) registering the parties and (ii) transforming SLAs into smart contracts. An excerpt of the respective code, in Solidity, is shown below. Line 1 defines a smart contract, while lines 2-5 introduce the variables concerned with the identification of the involved consumers and providers as well as with the definition of the contract states, each composed of a set of terms (the definition of the terms is omitted for the sake of simplicity). Line 7 defines an event to log the creation of a service smart contract, while lines 9-12 introduce a function for actually instantiating new smart contracts that receive as argument the SLAC SLA, creates the service smart contract (SSC), saves its references and emits the previously defined event.

```

1  contract ServiceSC {
2  address[] public consumers;
3  address[] public providers;
4  mapping(uint => Term []) states;
5  ... }
6  event sSCCreated(address _contract);
7  function newSSC (struct SLA) internal {
8  address sSC = new SSC(SLA);
9  sSCs . push (sSC);
10 sSCCreated (sSC) }

```

The dynamic part of the agreement is verified before the respective guarantees. If the predefined conditions hold, the state, that is, the valid terms of the agreement at that moment, is modified. To illustrate this concept, in the following listing, we report the dynamism section of a SLAC SLA; the reader is referred to Reference 2 for a complete account of SLAC. The rule encompasses the definition of the party, its type (eg, demand, request), the conditions under which to invoke an action and the definition of the envisaged actions. In particular, the rule specifies that upon consumer's demand (Alice, in this case), either the type of the offered VM is changed from small to large (vertical scaling scenario where its realization depends on the number of requests to the application component running on the offered VM) or the instances of the large VM will be increased by one (horizontal scaling scenario where its realization depends on whether the execution time of the user application component is above 100 seconds). On the other hand, the contract states that if the number of instances of the large VM type is greater than 1, then the availability of these instances will become 97%. This covers the provider side, which might not be able to sustain the same availability level when it has to maintain a large number of large VM instances. Please note that the considered SLA contains also other sections, like the static one, which corresponds to the initial state of

⁵<http://solidity.readthedocs.io/en/v0.4.24/>

the contract, describing the original SLA guarantees that need to be delivered. Additional details about this structure can be found in References 2 and 7.

```

1 on Alice demand:
2   if requestNumber > 100
3     replace SmallVM with LargeVM
4   else if executionTime > 10
5     replace value of LargeVM with old+1
6 on IMT authorization request:
7   if LargeVM > 1:
8     replace value of LargeVM.Availability with 97

```

The next listing contains a short excerpt of the smart contract that executes these modifications. The implementation covers cases where the parties (i) ask for a modification (it will be automatically executed without the need of authorization of other parties) or (ii) request authorization from other parties (it will be granted only if accepted by the inquired party). In case of requests, we first verify if the requester is within the requested rule's white-list, which is generated according to the definitions of the SLA. Then, in the terms of the agreement that require the authorization of the counterparts (line 4), the service is modified only if all involved parties agree and everyone is notified of the outcome of the request. In case the action is a demand, the service is automatically modified by sending the demand directly to the service provider and by notifying the involved parties (by calling the *modifyTerm* function).

```

1 function modifyTerm(uint ruleID , Action action) public returns (bool){
2   if (whitelist[ruleID] == msg.sender){
3     if (rules[ruleID] == request){
4       requestModification(ruleID , action);
5     } else{
6       modifyTerm(ruleID , action)
7     }
8   ...

```

4 | CROWD-BASED ORACLE-AS-A-SERVICE FOR CONSENSUS ON QOS MONITORING

A major limitation of blockchain technology is its inability to interact effectively with the outside world. Actual solutions are based on using a trusted third party, termed as the “oracle.” However, the centralized concept of an oracle is to some extent not completely reliable. Centralized oracles capture data directly from third-party service providers, websites, or different types of APIs and then report the results to smart contracts; this makes it possible to manipulate the results. In addition, the oracle is a SPoF as both monitoring and results validation depend on a single entity. Indeed, in the case of service monitoring, if the oracle fails, service QoS cannot be verified. A possible solution to increase credibility and resilience is to rely on multiple distributed oracles. However, since different oracles capture data from the outside world at the same time, a protocol is necessary to reach consensus about the collected data.

4.1 | COaaS framework architecture

In this section, we propose a framework, named crowd-based oracle-as-a-service (COaaS), which realizes the second level of our proposed architecture, to enable the creation of decentralized oracle solutions based on a customizable number of oracles that do reach consensus on the outcomes of QoS monitoring. The framework addresses some of the main limitations in the interaction between the real world and the blockchain: (i) the need for trusted third parties, (ii) the lack of transparency, and (iii) a SPoF, when relying on a single oracle. In our framework, a blockchain is leveraged to improve the reliability of the process of reaching a consensus among distributed oracles. Figure 4 shows the architecture of the COaaS framework.

First, we introduce a smart contract for managing all the oracles implemented on the transaction network, which is named “crowd oracle manager” (COM). The functionalities of this COM smart contract include:

- Individual oracles can register in the COM to become members of the crowd oracle pool. Registration of an oracle has to be authenticated since those oracles will form a decentralized monitoring network for the transaction network services.

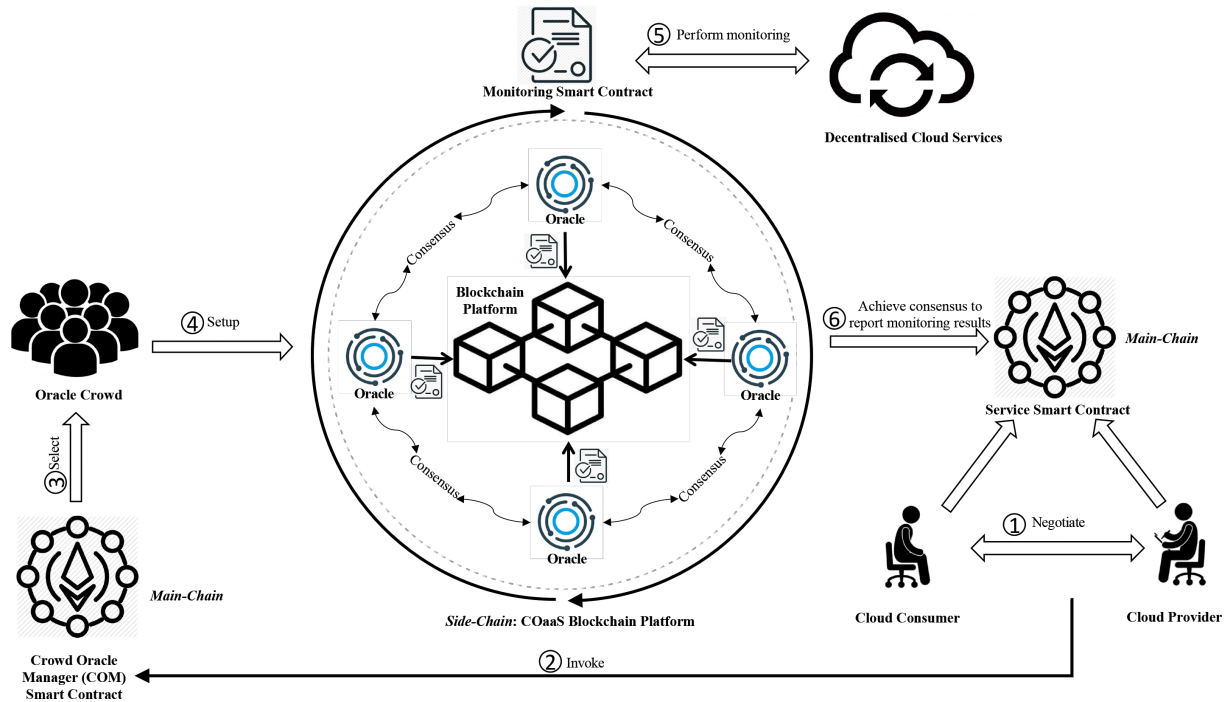


FIGURE 4 System architecture for COaaS framework

- After registration, the oracle can claim a public key in the COM smart contract that can be used in the side chain. This allows the service smart contract to acknowledge only those transaction data from the side chain, which contains the corresponding signatures of the selected oracles.
- The COM smart contract provides an interface to select oracles for a specific service smart contract that implements an unbiased sortition algorithm, that is, an algorithm that ensures that the selection result is considered fair by both providers and consumers. To this end, two alternative algorithms are described in our recent work.^{23,24}
- The selected oracle can also get rewards (tokens) from the COM smart contract for performing service monitoring. These rewards can come from a deposit paid by the provider and the consumer through the service smart contract.

There are also reputation issues to manage oracles. Due to space limitations, we shall not analyze in detail the content of the COM smart contract; such details can be found from the witness-pool smart contract implementation in our previous work.²⁴ The entire COaaS lifecycle, described in Figure 4, consists of the following steps:

1. The cloud provider and consumer negotiate the service provision strategy and generate a specific SLA/service smart contract. This is achieved using the SLA2SC component mentioned in Section 3.
2. The created service smart contract is executed and can invoke the selection interface of the COM smart contract when the respective need arises (eg, the service is deployed and ready to be monitored).
3. The unbiased sortition algorithm of the COM smart contract selects the individual oracles. The selection result is returned as a list in the service smart contract. This is essential to ensure that the service smart contract only admits transaction data from the selected oracles.
4. Each oracle periodically checks its status in the COM smart contract. As such, it can be notified about its selection, since the COM smart contract is deployed on the permissionless blockchain, where all information on the chain is public. Each oracle can also be informed about the other selected oracles and about the smart contract of the (cloud) service. All selected oracles create an oracle federation, which is deployed on the underlying blockchain platform, called the side-chain. Each oracle can join the blockchain network using cloud resources like VMs and a monitoring smart contract is deployed on the side chain.
5. The service smart contract notifies each oracle from the obtained list to start performing service monitoring and submitting transactions to the monitoring smart contract. The latter coordinates the consensus about SLO violation. The detailed process is that each oracle monitors the service and submits any violation detection result as a transaction to the monitoring smart contract. The transaction is then validated by other participating oracles based on their own monitoring information. The final validation result is determined by the specific consensus algorithm adopted by the permissioned blockchain platform.
6. Only validated transactions are recorded and stored on the blockchain. Since an SLO violation is not a common event, the transaction indicating the SLO violation will be reported to the service smart contract on the transaction network. Any oracle can transmit this transaction data to the

service smart contract, since the data are immutable due to the use of signatures and the service smart contract is, thus, able to verify it. In this case, the status of the service smart contract turns to “violated” and will not accept similar reports for the same SLO violation from other oracles. Meanwhile, the compensation is automatically enforced by the service smart contract.

It is worth mentioning that the service smart contract employs a time-window-based interface that is responsible for receiving SLO violation reports. Within the fixed time window employed for the measurement of an SLO's metric, the same SLO violation report would not be accepted twice. Afterwards, once the fixed time limit has passed, the SLO violation report can be accepted again, in order to count, for compensation purposes, the number of times an SLO has been violated. The time-span of the window should be designed to be shorter than the monitoring interval of the metric involved in the SLO. Hence, the duplicated SLO violations would not incorrectly increase the counter.

Through the architecture described above, the oracles in COaaS can offer traditional oracle functionalities while they guarantee the continuous monitoring and validation of dynamic cloud services in a cooperative manner.

4.2 | COaaS blockchain platform

Permissionless blockchains suffer from the problem of limited transaction throughput and scalability. On the other hand, permissioned blockchains address these problems by adopting more efficient consensus algorithms, such as practical byzantine fault tolerance (PBFT),²⁵ Proof of elapsed time (PoET),²⁶ and Raft.²⁷ This type of blockchains is usually not open and requires participants to be authenticated. Thus, they are more suitable for industrial applications in cloud computing and IoT scenarios, where there already exist basic authentication mechanisms among processing units and collaborating organizations.

In the COaaS framework, QoS monitoring is achieved through an underlying blockchain platform (side chain) that can take a different form (permissionless or permissioned) depending on the respective scalability, stability, and performance requirements. We have targeted a permissioned blockchain implementation for our underlying platform in the COaaS framework because of the following rationale:

- QoS measurement accuracy: While the consensus algorithms for confirming new transactions in permissionless blockchains are nondeterministic (eg, proof-of-work), in permissioned blockchains, the results are often deterministic.²⁸ Nondeterministic consensus algorithms rely on difficult problems (like finding a specific hash value) that have to be solved by the involved parties for safety reasons. Solving such problems requires a considerable amount of time. For example, Ethereum takes about 15 seconds to commit the latest submitted transactions, while Bitcoin takes around 10 minutes.²⁹ This is the main limitation for blockchain-based cloud monitoring, in particular for the storage of cloud monitoring data since high QoS measurement accuracy cannot be achieved (eg, service violations cannot be detected in 5-second windows when the time to confirm transactions of the underlying blockchain is higher than that). Permissioned blockchains confirm new transactions much faster.
- Oracle identity authentication: The transaction confirmation process on permissioned blockchains is more secure. Oracles joining the network must be authorized to read, write, or audit the blockchain data. Moreover, permissions can be associated with different access levels and the monitoring data can be encrypted.
- QoS monitoring cost: Permissionless blockchains are often more energy consuming than permissioned blockchains. In addition, in a permissionless blockchain, some tokens (eg, bitcoin or ether) are required to reward participants of the network that help to confirm transactions with *transaction fees*. According to our design, all the monitoring information is submitted as transactions. The oracle would then pay a large number of tokens for transaction fees when using a permissionless blockchain. The required transaction fee precludes the submission and storage of large amounts of data coming from continuous monitoring. In a permissioned blockchain platform, instead, all transactions can be submitted without transaction fees, which means that long-term, large-scale cloud monitoring is possible.

Since we aim at providing insights in the appropriateness of permissioned blockchain-based clouds and not to measure or compare specific tools, we adopted one of the most popular open source permissioned blockchain solution, the Hyperledger Sawtooth⁶ platform, which provides different permissioned blockchain platforms, including Fabric⁷ and Sawtooth⁸. Among these platforms, the architecture of Sawtooth is the most modular and contains all the typical components of a blockchain, such as the consensus mechanism, the validator, and the transaction processor (smart contract).³⁰ In addition, Sawtooth can support pluggable consensus algorithms; especially, the PoET consensus algorithm provided by Sawtooth appears to be scalable and able to improve performance. Thus, we adopt Sawtooth with PoET consensus for our prototype development and validation.

⁶<https://www.hyperledger.org/>

⁷<https://github.com/hyperledger/fabric>

⁸<https://github.com/hyperledger/sawtooth-core>

4.2.1 | COaaS prototype implementation

Before describing the detailed monitoring and transaction validation process, we briefly introduce the smart contract execution mechanism in Sawtooth. Each Sawtooth node maintains a key-value structure as a state dictionary, where a key contains 70 characters. Every type of smart contract needs to define a namespace to find its corresponding values through the mappings of the state dictionary. The transaction payload identifies the operation and input value for executing the smart contract. The validator component of Sawtooth receives the transaction and forwards it to the respective transaction processor. The transaction processor contains the smart contract and executes the transaction according to its payload data. Here, the transaction is first verified; if it is validated and consensus is reached, the corresponding values in the state dictionary are changed.

According to the smart contract programming example of Sawtooth⁹, the basic logic of a COaaS smart contract for service violation detection and validation can be defined considering the following aspects:

State

The state value for a COaaS smart contract only requires one value to indicate whether the corresponding SLO is violated or not. Therefore, we define the state value as a Boolean value.

```
<'violation'>
```

Addressing

The provider and consumer negotiate a name, "xName," as the namespace for a specific smart contract. In addition, there should be an integer value, "SLO," provided by the transactions to indicate the corresponding SLO negotiated by the provider and consumer. Then, the state value can be found through hash mapping. The following is a python example of how to perform the hash mapping to find the corresponding state value.

```
>>> hashlib.sha512('xName'.encode('utf-8')).hexdigest()[ :6]
+hashlib.sha512('SLO'.encode('utf-8')).hexdigest()[ -64:]
```

Transaction payload

The transaction request's payload of a COaaS smart contract contains two values: an integer value "SLO" to indicate the SLO index and a Boolean value, "result" to indicate the corresponding state value named as "violation."

```
<'SLO'><'result'>
```

Execution

The COaaS transaction processor, which is programmed for executing the COaaS smart contract, receives a transaction request. The transaction is identified as invalid when one of the following three conditions holds.

- the payload of the transaction request is empty;
- the payload does not include a valid integer value (ie, a mapping to an existing SLO) and a valid Boolean value;
- the monitoring information of the "SLO" is in contrast with the payload value named as "result."

After receiving a transaction request, the oracles communicate through a specific consensus algorithm to achieve the final validation result. When using the PBFT consensus algorithm, there are three steps: in the preparation phase, an oracle submits the transaction to other oracles; in the preparation phase, the other oracles communicate their validation results for the transaction; and in the last commit phase, all the oracles take their decision based on the results received in the previous phase and notify their conclusions to the others. After receiving the decisions of the others, each oracle is able to determine whether the transaction is valid and can thus be committed in the blockchain, with the corresponding state value, "violation," set according to the "result."

Figure 5 depicts the sequence diagram showing the detailed process of COaaS. During the phase of COaaS deployment, cloud providers and consumers need first to negotiate; after the successful negotiation of the terms, the service smart contract is created in the transaction network and executed. Upon this contract's execution, when the service is deployed and ready to be monitored, the selection interface of the COM smart contract (also implemented in the transaction network) can be invoked. Then, the unbiased sortition algorithm implemented in the selection interface selects several oracles from a trustworthy oracle pool to form the side chain. The selected oracles, once notified, download the corresponding monitor

⁹https://sawtooth.hyperledger.org/docs/core/releases/1.0.1/transaction_family_specifications/integerkey_transaction_family.html

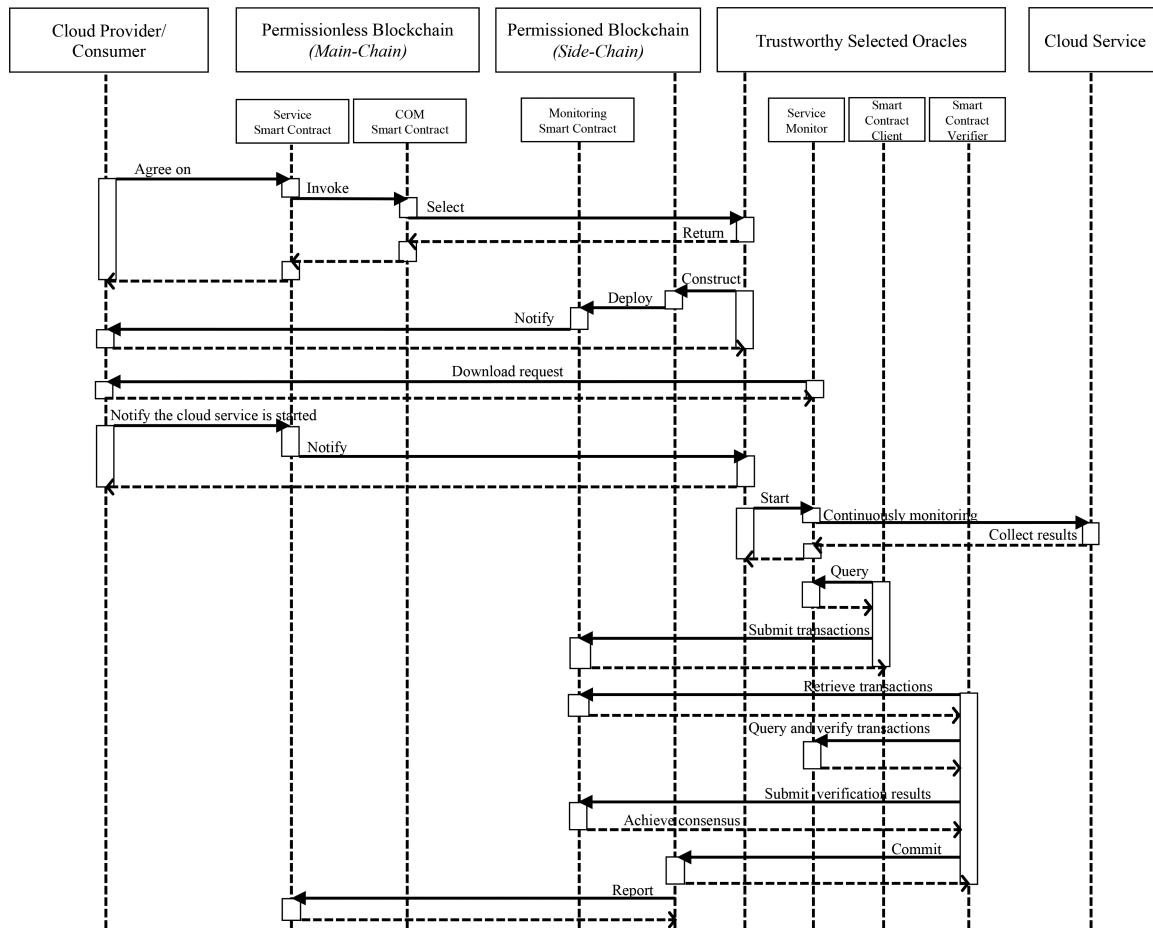


FIGURE 5 Sequence diagram of the COaaS framework

to perform the service monitoring. Since different services have specific features and promise different SLOs in their smart contract, the service monitor should be provided by the provider and the consumer, who both should agree about its implementation. In addition, the service monitor can be wrapped as a container for easy deployment and execution. The outcome of the service monitor's deployment and execution is the indication of SLO violations.

In the runtime phase of COaaS, cloud providers and consumers change the SLA status to be considered and the corresponding oracles are notified to start their monitoring service. The decentralized oracles leverage the service monitor to individually monitor the cloud service, and the smart contract periodically query the service monitor, and submit transactions, which, based on the monitoring results, indicate to the permissioned blockchain whether the SLO has been violated. The smart contract verifier, that is, the COaaS transaction processor, of all the oracles, validates each newly submitted transaction. For validation, the verifier queries its own service monitor to verify the transaction based on the monitoring result. Based on their own validation, all verifiers further communicate with each other to reach consensus. When the transaction is validated with the consensus of all verifiers, the final validation result is committed on the side chain. If the SLO violation is confirmed, the committed transaction is transmitted to the transaction network's service smart contract to trigger the SLA enforcement.

5 | VALIDATION

In this section, we discuss the experiments related to performance, scalability, and stability of our permissioned blockchain implementation.

5.1 | Performance validation for orchestrating the COaaS blockchain platform

According to the sequence diagram of the COaaS framework shown in Figure 5, each oracle needs computing resources to operate service monitoring, smart contract, and verifier. Specifically, all oracles have to communicate to form a COaaS blockchain network. Our blockchain platform

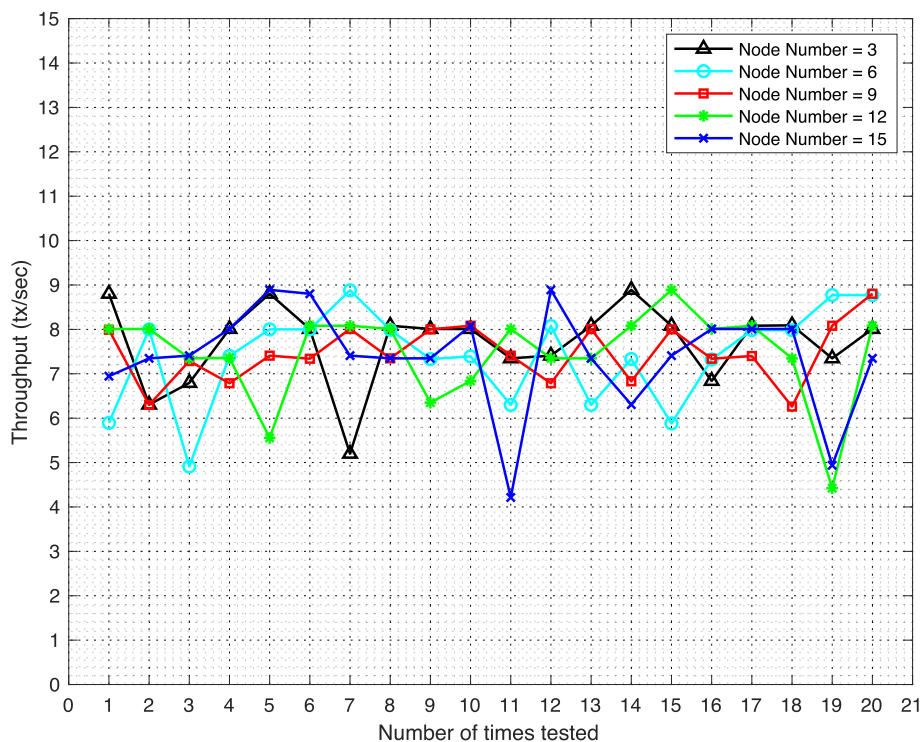


FIGURE 6 Variation in performance of different VM node numbers

relies on the Sawtooth implementation. In essence, the cloud environment is suitable for operating this blockchain service as it can offer elastic VM and network resources. Here, we evaluate performances when the Sawtooth blockchain is used as the underlying platform of the COaaS framework. Through the experimental study, we aim at getting insights about the COaaS platform's orchestration and about the relevant aspects to consider.

First, we would like to describe basic assumptions and settings of the experiment.

- In principle, each oracle can choose the cloud resources to form the blockchain platform. However, to control variability, we simulate a scenario where all blockchain nodes are deployed in one cloud data center where each oracle leverages one VM to be one of the blockchain nodes in the blockchain platform.
- For the consensus algorithm, we adopt the PoET algorithm, which appears to be scalable.³¹
- We choose the IntKey smart contract provided by Sawtooth as the workload input. The workload contains a bunch of transactions that can be submitted to the blockchain platform according to a specific rate.
- The functionality of IntKey smart contract includes setting a value by increasing or decreasing the value by one. Hence, the execution process for an IntKey transaction has also to check and change a value in the state dictionary. This means that the complexity of the IntKey smart contract processor is similar to that of the COaaS transaction processor described in Section 4.2.1. Thus, the IntKey workload can impact the performance of a COaaS-type smart contract.
- We use two clouds as test beds: AWS¹⁰ and ExoGENI¹¹, and, for each of them, we use six data centers and three VM types.

5.1.1 | Impact of VM node numbers

In this experiment, we keep the input transaction rate (9 tps) of the workload and adjust the number of VM nodes to evaluate the throughput performance of our COaaS blockchain. Figure 6 and Table 1 show that the throughput performance stays stable with a different number of VM nodes. In particular, there is no significant throughput decrease when the number of VM nodes increases. This indicates a good scalability level for the PoET consensus algorithm, since the performance of a traditional consensus algorithm, for example, PBFT, would dramatically decrease as the node number increases. Hence, the PoET consensus algorithm can be instrumental when a large number of oracles is required.

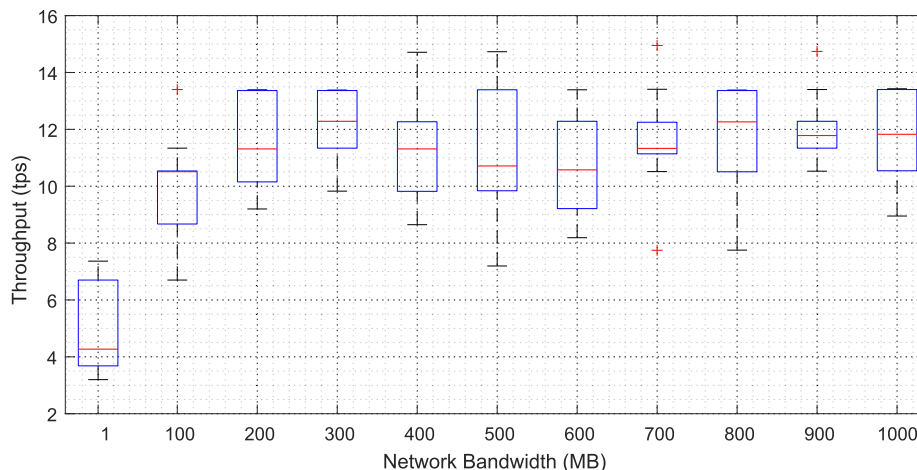
¹⁰<https://aws.amazon.com/>

¹¹<http://www.ExoGENI.net/>

TABLE 1 Performance variation in terms of different number of VMs

Number of VMs	Average throughput (tps)	Average duration (s)	Variance
3	7.75	116.60	16.34
6	7.43	122.20	20.10
9	7.47	119.80	10.58
12	7.46	122.05	22.23
15	7.40	124.00	26.13

FIGURE 7 Variation in performance in terms of different network bandwidths



5.1.2 | Impact of network conditions

To evaluate the throughput of our COaaS blockchain, we keep fixed the input transaction rate (15 tps) of the workload and adjust the bandwidth between the VM nodes. Figure 7 shows that when the network bandwidth is greater than 100 MB, the median of throughput is stable and kept around 12 tps. This indicates that platform performance is stable irrespective of the bandwidth, while there are only a few cases where 1 or 2 outliers occur across different bandwidths. When we limit the bandwidth to 100 MB, the performance starts exhibiting a dropping trend. If the bandwidth drops to 1 MB, the platform performance drops to around one-third (4 tps). From the above results, we can conclude that the bandwidth can be a bottleneck for the COaaS blockchain. Hence, when building the COaaS blockchain, the bandwidth among nodes should be kept above a specific threshold, 100 MB in our case. Furthermore, in current cloud systems, the maximum bandwidth is also affected by the VM type¹². It is, therefore, essential to use an appropriate VM type to attain a specific bandwidth level. Finally, it has been observed that better bandwidths can be achieved if VMs are placed at the same location. Thus, this can be another recommendation for guaranteeing a specific bandwidth target.

5.1.3 | Impact of VM types

We use the workload with a large input transaction rate (50 tps) to ensure that the platform performance always reaches the bottleneck, no matter what types of VMs are leveraged. Our objective is to evaluate the COaaS blockchain performance when adjusting the VM types with various capabilities. Figure 8 shows the performance analysis results with different VM configurations, with the first three mapping to the ExoGENI provider and the last three to AWS. Overall, in our experiments, AWS outperforms ExoGENI as it offers better VM capabilities for similar VM types. The reason is probably that even with the same configuration (number of vCPUs, main memory, and disk size), different clouds attain a diversified performance level. Notably, it can be observed, for both ExoGENI and EC2, that as the VM type changes from small, to medium to large, the average performance (throughput median) of the COaaS blockchain improves in a normal or quite significant degree depending on the specific cloud involved. Hence, when constructing the COaaS blockchain, the VM (instance) type with different capabilities should be considered. Using a VM with better capabilities can lead to higher performance, but there is obviously a trade-off with monetary cost.

¹²<https://cloudonaut.io/ec2-network-performance-cheat-sheet/>

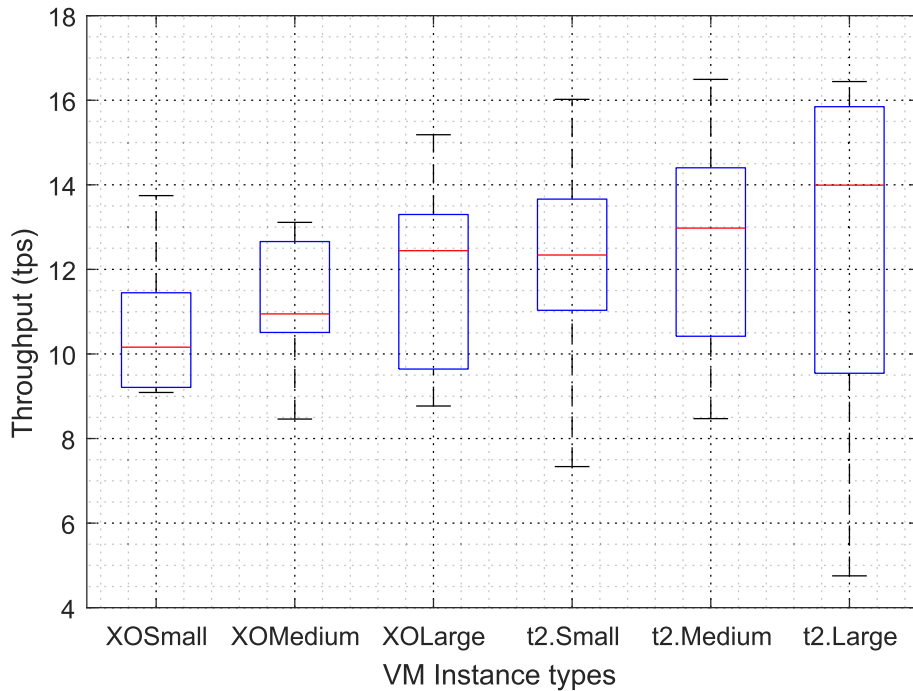


FIGURE 8 Variation in performance of different VM types

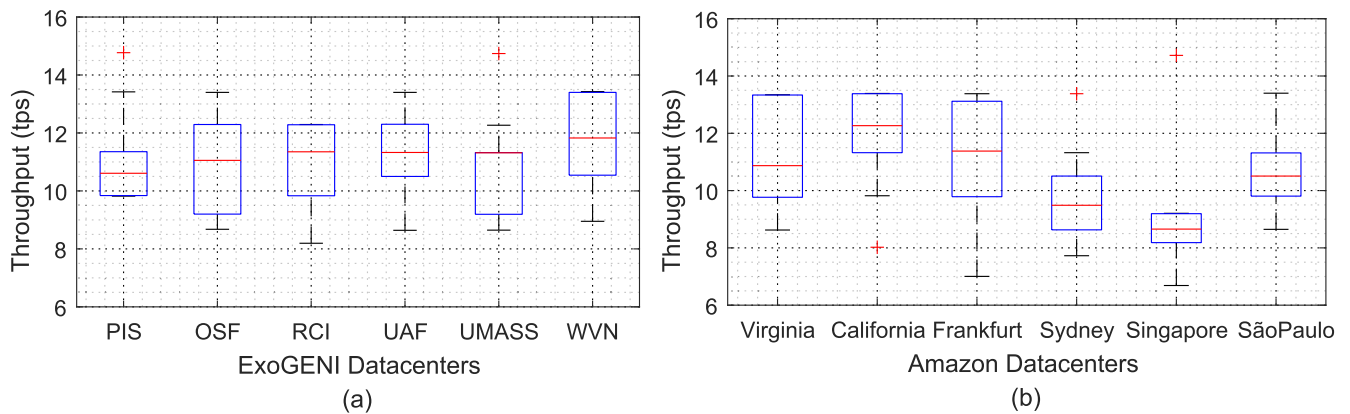


FIGURE 9 Variation in performance of different cloud providers across various data centers

5.1.4 | Impact of data centers

We test whether a data center can influence performance when the same VM type is used while we make sure that the VMs of both clouds have a similar capacity by using as a VM type “XOSmall” for ExoGENI and “t2.Small” for Amazon. Figure 9 considers the six ExoGENI data centers, all located in the United States. Among them, the performance of UAF and PIS is more stable than that of the others, while WVN has the highest average throughput. We checked resource utilization of different data centers (which is possible with ExoGENI) and found that lower utilization always resulted in higher performance. When many consumers use the same cloud resource at the same time, the data center becomes busy and the actual performance worsens due to shared resources contention (see, eg, UMASS in Figure 9). On the contrary, when the data center is relatively idle, the performance of the Sawtooth platform improves (see, eg, WVN). In the AWS section, the California rack has the highest and quite stable performance with an average throughput over 12 tps, followed by the data centers of Frankfurt and Virginia. By contrast, the throughput of Australia and Singapore data centers is lower. An interesting observation is that although Singapore has the lowest (throughput) performance, its throughput is quite stable and maintained between 8 and 10 tps. Hence, when constructing the COaaS blockchain, the selection of cloud and data center can influence the performance. A potential reason is the utilization difference among the data centers. If utilization information is provided (like in ExoGENI), one could select the relatively idle data centers; otherwise, the selection to achieve better performance can only be based on real-time profiling (like in EC2).

5.1.5 | Lessons learnt

Figure 4 shows that the blockchain platform is the main component of the COaaS framework. Therefore, the blockchain infrastructure can directly affect the efficiency of the COaaS framework when performing service monitoring and SLO violation detection, and thus can influence the service smart contract enforcement in the transaction network.

Considering the trend of leveraging clouds to operate a permissioned blockchain service, we conducted the performance validation with Sawtooth to simulate different scenarios according to our COaaS framework. The main findings are the following:

- The number of oracles used in the COaaS blockchain affects the performance for consensus attainment reasons. We demonstrate that the PoET consensus algorithm guarantees a good scalability level.
- The bandwidth among the nodes of the COaaS blockchain affects performance within a certain threshold. Hence, the bandwidth should be properly customized. Very high bandwidth values are of no help for further improving performance.
- The VM type with a bigger capacity can enhance performance. Thus, an appropriate VM type needs to be considered to achieve a balance between performance and costs.
- The data center where the COaaS blockchain platform is deployed also impacts the platform performance. According to our experiments, an idle data center with low utilization can enable to reach a higher performance level. Nevertheless, a more practical solution is achieved by leveraging real-time profiling before selecting the data center to perform the deployment.

6 | DISCUSSION

Today, the cloud market is restricted to a very small number of large providers; this entails higher prices, vendor lock-in, and lower service quality. Smart contracts and blockchains have the potential to reshape this market and boost the adoption of clouds; thus, improving QoS and reducing costs and prices. Smart contracts and blockchains can also contribute to considerably reducing the potential environmental cost of the blockchain^{32,33} by using appropriate consensus algorithms, like the one proposed in this article.

Moreover, supporting QoS terms, for example, response time and jitter, and considering the distributed nature of the system that gathers geographically sparse micro and traditional clouds will leverage edge computing and enable real-world use of this new paradigm for next-generation cloud applications. This scenario and a higher adoption rate can only be achieved by increasing the trust of cloud consumers. Such a trust is currently affected by (i) the supply of contracts in a take-it-or-leave-it basis with limited QoS guarantees; (ii) the static nature of QoS guarantees, which is not suitable in the context of the dynamic environment under which cloud services operate; (iii) the centralized and thus untrusted SLA management via SLA tools and techniques, which are supplied exclusively by the cloud service providers.

To address the above challenges and to pave the way to a really open and flexible cloud market, this article has proposed a novel framework, which relies on (a) dynamic SLAs that cover a rich set of QoS guarantees and the evolution of such guarantees over time; (b) a two-level architecture of blockchains by means of which dynamic SLAs can be managed in a distributed manner through smart contracts: (i) at the top level, a public blockchain handles the execution of the smart contract and enables any entity to participate in the open market and (ii) at the lower level, a side-chain handles the production of objective measurements for SLA evaluation purposes through the exploitation of a carefully selected network of oracles.

A similar architecture can also be used by a single provider to lower the market entry barrier by offering consumers some independent mechanisms, that is, a smart contract and blockchain, for the verification and enforcement of service quality and distributed SLA management. In this case, the main difference with the open marketplace is that consumers are rewarded for participating in services verification.

The envisioned framework has been realized by extending our previous work^{2,8,20} concerned with a language and an associated framework for the definition and management of dynamic SLAs, with the capability of (i) transforming dynamic SLAs into smart contracts, (ii) enforcing smart contracts via the blockchain by relying on appropriate mechanisms for contract management and consensus attainment, and (iii) exploiting an oracle interface for retrieving information about the sensed cloud service for better supporting decision making during the SLA enforcement.

To realize the second level of the proposed architecture, we have implemented a crowd-based oracle for overcoming one of the main challenges in the smart SLA management: the collection, validation, and insertion of objective monitoring data in the blockchain. The proposed solution, at its highest level, is based on the public chain; however, the public blockchain does not scale well and has prohibitive costs for applications that require many transactions and produce a multitude of monitoring data; we have, thus, used a side-chain network. We acknowledge that relying only on a permissioned side-chained environment and avoiding the public blockchain would simplify our architecture, but it would not be aligned with our aim to foster open cloud markets without privileged roles and with low entrance barriers for providers and users.

In order to avoid single point of failure and to avoid relying on trusted third parties, we have developed a distributed oracle in a side chain, that is, another blockchain network with permissioned access. To measure the actual performance of the developed solutions and the underlying blockchain

(based on the hyperledger Sawtooth), we have conducted experiments in different settings and with different cloud providers. The results highlight that the performance of this second-level blockchain depends on many factors, including the network conditions of the cloud service provider and the resource utilization of the respective data centers. However, if the different cluster configurations are appropriately tuned, performance becomes insensitive to network bandwidth. Even if at a first glance the performance of the permissioned blockchain (12 transactions per second) might appear to be a bottleneck for a large network, our framework has been designed in such a way that for every new service a new permissioned blockchain is created among selected oracles. Therefore, full capacity is dedicated to a single service, which is more than enough in most of the cases.

Overall, our approach could be extended and improved in many ways:

- The expressive power of smart contracts brings flexibility to QoS definition, enabling users to model different scenarios, and to cover also future needs of both signatory parties. However, matching^{34,35} and negotiating³⁶ offers and requests, with such a wide range of possibilities, are challenging. Hence, we plan to support online multistate service matching and improve scalability of cloud service negotiation that is currently performed offline with no scalability guarantees. Both capabilities could be a competitive offer in the portfolio of a cloud broker. Moreover, the system could verify the potential interference among services by the same provider to improve the matchmaking and reduce risk of SLA violations, for example, by using the approach described in Reference 37.
- Static analysis is currently carried out on the SLAs before they are transformed into smart contracts. We plan to support the transformation into probabilistic automata to guarantee formally verified properties related to, for example, reachability and liveness, and to provide probabilistic proofs of the equivalence between the defined SLA and the generated smart contract.
- The current decentralized monitoring solution can be improved by producing high-level measurements, on top of the objective raw ones, by relying on statistical analysis and by supporting an informed selection of the oracles according to their reputation. For the actual selection of oracles, we plan to experiment with different consensus protocols to assess their suitability under different circumstances.
- The current time-window based solution, used to filter duplicated service-level objectives violation reports, can be improved by empowering oracles with the ability to reach consensus on the violation to report; this handling of duplicated violations avoids considering them in the actual contract. Currently, we assume that each oracle is able to produce measurements associated with specific QoS parameters. However, we do not investigate the actual placement of the oracles³⁸ and the techniques they use to produce such measurements as they can depend on the monitored QoS parameter. We plan to investigate these issues in the near future.
- Privacy needs to be better considered;³⁹ in the proposed solution, all participants can read and execute the smart contracts and could learn about the identity of the parties and about some intrinsic details of the offered services.
- The direct and indirect cost of creating a permissioned blockchain for each service will have to be more precisely measured.
- The performance of other permissioned blockchain platforms, such as Fabric, should be assessed.

ACKNOWLEDGMENT

This work has been partially funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 838949, by the ARTICONF project grant agreement No 825134, by the ENVRI-FAIR project grant agreement No 824068, by the BLUECLOUD project grant agreement No 862409, by the LifeWatch ERIC, and by the MIUR project PRIN 2017FTXR7S IT-MATTERS (Methods and Tools for Trustworthy Smart Systems).

ORCID

Rafael Brundo Uriarte  <https://orcid.org/0000-0003-0750-7376>

Huan Zhou  <https://orcid.org/0000-0003-2319-4103>

Zhiming Zhao  <https://orcid.org/0000-0002-6717-9418>

REFERENCES

1. Opara-Martins J. Taxonomy of cloud lock-in challenges. *Mobile Computing - Technology and Applications*. London: IntechOpen; 2018.
2. Uriarte RB, De Nicola R, Scoca V, Tiezzi F. Defining and guaranteeing dynamic service levels in clouds. *Futur Gener Comput Syst*. 2019;99:27-40.
3. Bartoletti M, Pompianu L. An empirical analysis of smart contracts: platforms, applications, and design patterns. Paper presented at: Proceedings of the International Conference on Financial Cryptography and Data Security; 2017:494-509; Springer.
4. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system; 2008. <http://bitcoin.org/bitcoin.pdf>.
5. Buterin V. A next-generation smart contract and decentralized application platform. White Paper; 2014.
6. Uriarte RB, De Nicola R. Blockchain-based decentralised cloud/fog solutions: challenges, opportunities and standards. *IEEE Commun Stand Mag IEEE*. 2018;2(3):22-28.
7. Uriarte RB, Tiezzi F, De Nicola R. Dynamic SLAs for clouds. Paper presented at: Proceedings of the 5th IFIP WG 2.14 European Conference, ESOC 2016, Service-Oriented and Cloud Computing September 5-7, 2016, Proceedings; 2016:34-49; Springer; Vienna, Austria.

8. Uriarte RB, De Nicola R, Kritikos K. Towards distributed SLA management with smart contracts and blockchain. Paper presented at: Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom); 2018: 266-271; IEEE.
9. Crosby M, Pattanayak P, Verma S, Kalyanaraman V. Blockchain technology: beyond bitcoin. *Appl Innovat*. 2016;2(6-10):71.
10. Bartoletti M, Pompianu L. An empirical analysis of smart contracts: platforms, applications, and design patterns. Paper presented at: Proceedings of the International Conference on Financial Cryptography and Data Security; 2017:494-509; Springer.
11. Xu X, Weber I, Staples M. A taxonomy of blockchain-based systems for architecture design. Paper presented at: Proceedings of the IEEE International Conference on Software Architecture; 2017:243-252.
12. Zhang F, Cecchetti E, Croman K, Juels A, Shi E. Town crier: an authenticated data feed for smart contracts. Paper presented at: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; 2016: 270-282; ACM.
13. Ritzdorf H, Wüst K, Gervais A, Felley G, Capkun S. TLS-N: non-repudiation over TLS enabling ubiquitous content signing for disintermediation. *IACR ePrint Rep*. 2017;578:578-594.
14. Ellis S, Juels A, Nazarov S. ChainLink: a decentralized oracle network. White Paper; 2017.
15. Kritikos K, Pernici B, Plebani P, et al. A survey on service quality description. *ACM Comput Surv*. 2013;46(1):1:1-1:58.
16. Andrieux A, Czajkowski K, Dan A, et al. *Web Services Agreement Specification (WS-Agreement)*. Seattle, Washington: Open Grid Forum; 2007.
17. Kritikos K, Plexousakis D, Plebani P. Semantic SLAs for services with Q-SLA. *Proc Comput Sci*. 2016;97:24-33.
18. Serrano D. SLA guarantees for cloud services. *Futur Gener Comput Syst*. 2016;54:233-246.
19. Keller A, Ludwig H. The WSLA framework: specifying and monitoring service level agreements for web services. *J Netw Syst Manag*. Springer; 2003;11(1):57-81.
20. Uriarte RB, Tiezzi F, De Nicola R. Dynamic SLAs for clouds. In: Aiello M, Johnsen EB, Dustdar S, Georgievski I, eds. Paper presented at: Proceedings of the 5th IFIP WG 2.14 European Conference on service-oriented and cloud computing, ESOC 2016; September 5-7, 2016; 2016:34-49; Springer International Publishing, Vienna, Austria.
21. Nakashima H, Aoyama M. An automation method of SLA contract of web APIs and its platform based on blockchain concept. Paper presented at: Proceedings of the 2017 IEEE International Conference on Cognitive Computing (ICCC); 2017:32-39; IEEE.
22. Adler J, Berryhill R, Veneris A, Poulos Z, Veira N. Astraea: a decentralized blockchain oracle; Paper presented at: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData); 2018:1145-1152; IEEE.
23. Zhou H, Laat C, Zhao Z. Trustworthy cloud service level agreement enforcement with blockchain based smart contract. Paper presented at: Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom); 2018:255-260; IEEE.
24. Zhou H, Ouyang X, Ren Z, Su J, Laat C, Zhao Z. A blockchain based witness model for trustworthy cloud service level agreement enforcement. Paper presented at: Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications; 2019:1567-1575; IEEE.
25. Castro Miguel, Liskov Barbara. Practical byzantine fault tolerance. In: Seltzer Margo I, Leach Paul J., eds. Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI); 1999:173-186; USENIX Association.
26. Chen L, Xu L, Shah N, Gao Z, Lu Y, Shi W. On security analysis of proof-of-elapsed-time (poet). Paper presented at: Proceedings of the International Symposium on Stabilization, Safety, and Security of Distributed Systems; 2017:282-297; Springer.
27. Howard H. ARC: analysis of Raft consensus. No. UCAM-CL-TR-857. University of Cambridge, Computer Laboratory; 2014.
28. Cachin C. Architecture of the hyperledger blockchain fabric. Paper presented at: Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers; 2016.
29. Dennis R, Disso JP. An analysis into the scalability of bitcoin and ethereum. Paper presented at: Proceedings of the 3rd International Congress on Information and Communication Technology; 2019:619-627; Springer.
30. Olson Kelly, Bowman Mic, Mitchell James, Amundson Shawn, Middleton Dan, Montgomery Cian. Sawtooth: an introduction. The Linux Foundation; January 2018;.
31. Shi Z, Zhou H, Hu Y, Jayachander S, Laat C, Zhao Z. Operating permissioned blockchain in clouds: a performance study of hyperledger sawtooth. Paper presented at: Proceedings of the 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC); 2019:50-57; IEEE.
32. Truby J. Decarbonizing Bitcoin: law and policy choices for reducing the energy consumption of Blockchain technologies and digital currencies. *Energy Res Soc Sci*. 2018;44:399-410.
33. Zheng Z, Xie S, Dai H, Chen X, Wang H. An overview of blockchain technology: architecture, consensus, and future trends. Paper presented at: Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress); 2017:557-564; IEEE.
34. Klusch M. Semantic web service coordination. In: Schumacher M, Schuld H, Helin H, eds. *CASCOM: Intelligent Service Coordination in the Semantic Web*. Basel: Birkhäuser; 2008:59-104.
35. Rambold M, Kasinger H, Lautenbacher F, Bauer B. Towards autonomic service discovery a survey and comparison. SCC. Bangalore, India: IEEE Computer Society; 2009:192-201.
36. Marino F, Moiso C, Petracca M. Automatic contract negotiation, service discovery and mutual authentication solutions: a survey on the enabling technologies of the forthcoming IoT ecosystems. *Comput Netw*. 2019;148:176-195.
37. Uriarte RB, Tiezzi F, Tsaftaris SA. Supporting autonomic management of clouds: service clustering with random forest. *IEEE Trans Netw Serv Manag*. 2016;13(3):595-607.
38. Ke Y, Seshan S, Nath S, Karp B, Gibbons PB. IrisNet: an architecture for a worldwide sensor web. *IEEE Pervas Comput*. 2003;2(04):22-33.
39. Kosba A, Miller A, Shi E, Wen Z, Papamanthou C. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. Paper presented at: Proceedings of the IEEE Symposium on Security and Privacy (SP); 2016:839-858.

How to cite this article: Uriarte RB, Zhou H, Kritikos K, Shi Z, Zhao Z, De Nicola R. Distributed service-level agreement management with smart contracts and blockchain. *Concurrency Computat Pract Exper*. 2021;33:e5800. <https://doi.org/10.1002/cpe.5800>