Reliability Management for Blockchain-Based Decentralized Multi-Cloud

Atakan Aral
TU Wien
Vienna, Austria
atakan.aral@tuwien.ac.at

Rafael Brundo Uriarte
TU Wien
Vienna, Austria
rafael.uriarte@tuwien.ac.at

Anthony Simonet-Boulogne
iExec Blockchain Tech
Lyon, France
asb@iex.ec

Ivona Brandic
TU Wien
Vienna, Austria
ivona.brandic@tuwien.ac.at

***Abstract*—Blockchain-based decentralized multi-cloud has the potential to reduce cloud infrastructure costs and to enable geographically distributed providers of any size to monetize their computational resources. In this context, guarantees that the computational results are delivered within the promised time and budget must be provided despite the limited information available about the location and ownership of resources. Providers might claim to execute the services to get compensated for the computation even though returning incomplete or incorrect results. In this paper, we define a model to predict provider reliability, that is, the probability of failure-free execution of computational tasks and correctness of the computed outputs, by extracting the potential dependencies between providers from historical log traces. This model can then be utilized in the definition of provider reputation or the scheduling of new services. Indeed, we propose a probabilistic scheduler that chooses the providers that meet the reliability constraints among others. Finally, we validate the proposed solutions with real traces from a decentralized cloud provider and hint at the benefits of predicting reliability in this context.**

***Keywords*-Cloud Computing; Reliability; Smart Contracts; Distributed Ledgers; Blockchain; Distributed Management**

I. Introduction

Blockchain-based decentralized multi-cloud (BDMC) [1], [2], [3] combines smart contracts and blockchain to solve critical issues of traditional clouds, such as partial global coverage, vendor lock-in, and limited service offerings [4]. Service providers offer cloud services in the multi-cloud environment, consumers search and contract with these services, and the transactions are registered in the blockchain. These platforms automate the whole service life cycle, including the verification of the result correctness, which is regulated by consensus protocols based on the majority of votes. The role of the platform is critical in such scenarios, as small providers and fog nodes are especially prone to failures, since they are geographically dispersed, ad-hoc, low capacity, and lacking advanced support systems [5]. Additionally, BDMC presents the risks of Byzantine failures and malicious behavior, including servers claiming to run the tasks but returning incomplete or incorrect results to get compensated without fully providing the service.

In this context, reliability is a particularly important requirement as consumers have little or no knowledge of the resource location and ownership. Here, reliability is defined as the failure-free execution of computation tasks and the correctness of computed outputs. Due to the novelty of the underlying technologies, however, there are several research challenges related to reliability. The first challenge is how to define the desired level of reliability by the consumer in a smart contract. Availability, which is usually communicated through downtime, number of nines, or minimum service level, and reputation, is handled differently by the existing solutions [1]. In an attempt to standardize and simplify the terminology and to account for uncertainty, we propose a probabilistic approach for representing reliability in BDMC.

Another challenge is the accurate forecasting of the future reliability of a provider. Our previous work already showed that failures in geographically distributed computation nodes show a certain level of correlation [5]. In current work, we also hypothesize that the correctness of the results provided by seemingly independent providers can be correlated due to joint ownership. Moreover, such providers may decide to act maliciously only when they believe they have the numbers to impact the consensus for a particular task. To this end, our approach extracts the dependencies between the providers from historical log traces and incorporate them into our forecasting model. Finally, we also focus on the decision problem of choosing the providers that meet reliability requirements to schedule tasks. In summary, the following are our main contributions of this work.

- Three Bayesian network (BN) models for the reliability of providers representing the correlations between them with corresponding uncertainty (Sec. III),
- A reliability predictor for the services that are executed by multiple providers based on the models (Sec. III),
- A task scheduling algorithm that exploits redundancy to give probabilistic guarantees for correctness (Sec. IV),
- Extensive evaluation using traces from a BDMC platform, iExec [6], and insights (Sec. V and VII).

The proposed computation verification mechanism is novel in the following aspects: (i) it does not assume that BDMC providers are independent and uncoordinated; (ii) its reliability evaluation is preemptive, which brings cost and response time reductions; and (iii) it distinguishes inadvertent failures from malicious actions. A comparison to the previous work is presented in Sec. II and Sec. VI.

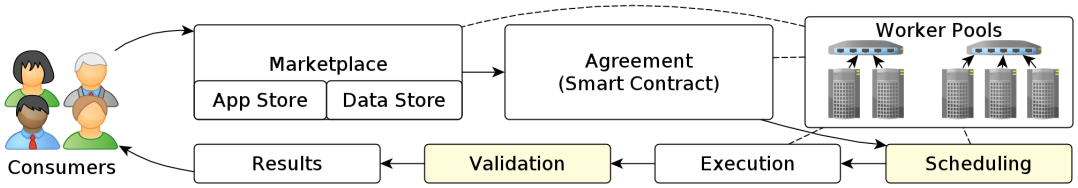


Figure 1. Decentralized Multi-Cloud Architecture. This Paper Proposes Improvements to the Scheduling and Validation Processes (Highlighted in Yellow).

II. Decentralized Multi-Cloud Environments

The Bitcoin and Ethereum boom and the recent developments in other distributed ledgers and smart contracts facilitated the security and trust in payments as well as verification of results. The blockchain [7] is an implementation of the distributed ledgers concept, which securely registers transactions, even in completely decentralized systems, and removes the need for trusted intermediaries; the only trusted element being the distributed and verifiable computing system. Smart Contracts are self-executing digital agreements between the provider and the consumer, which are flexible enough to represent various requirements and are consistently executed by a network of mutually distrusted nodes, without the arbitration of a trusted authority. Together these solutions encouraged the development of various decentralized solutions independent of individual providers [1].

In the cloud, the situation is similar and several ongoing projects attempt to create decentralized solutions for the cloud/fog market. In this context, BDMC could lower the infrastructure costs and bring predictable results by offering a purely decentralized system that manages the life cycle of cloud services and applications, does not depend on the market trust of individual partners, and paves the way for a truly open cloud market. Moreover, it enables geographically distributed providers of any size to offer computational resources, which could also boost fog computing offerings and adoption. In our previous work [1], we present a thorough analysis of these projects, discuss those benefits and challenges involved in building decentralized cloud/fog solutions and supply an overview of the applicable standards.

Fig. 1 depicts the high-level architecture of a typical BDMC platform. Here, Application and Data Providers create their applications/services/data and make them available in the *Marketplace*. The marketplace, normally defined in a smart-contract, contains the offers of computing resources time and the data sets and applications available in the BDMC. *Consumers* negotiate the terms of the service to access these applications and create a smart contract to manage the service provisioning. Then, the service is submitted to a *Worker Pools*, namely a group of workers managed by a *broker* in charge of offering to the consumers the available computing resources of workers, who execute the task according to the terms defined in the smart contract (e.g., response time and availability). The results are validated and, if correct, sent to the consumer. In this work, we focus on *Scheduling* and *Validation* processes by calculating the probabilities of failures or incorrect results.

Due to low throughput and the high processing costs in public blockchains, a BDMC has two different networks: the side-chain, also known as task network; and the main-chain or the transaction network [8]. The main-chain makes use of the blockchain to (more) safely register transactions, the smart contracts, regulate payments and reputation of the participants. The so-called *side-chain*, normally have restricted access (permissioned blockchain) and are publicly verifiable (e.g., transactions and smart contracts) by a smaller set of users. In the BDMC context, the side-chain manages all tasks that require processing by rewarding providers for computational resources, who communicate through a P2P network. It executes the actual services; marketplace that supports negotiation; and result verification module that verifies the service execution according to the specification.

However, the use of different networks raises several challenges, particularly because the side-chain is not publicly auditable. Smart contracts are designed for deterministic environments and trade-off immutability and trust for flexibility. Determinism is required by such contracts since a significant part of the network nodes executes them to achieve consensus (based on a specific protocol) over the fulfillment of the agreement terms. However, results of computation or monitoring information from the side-chain are non-deterministic and their use can lead to different results between the nodes executing the smart contracts and could break the consensus. Therefore, an interface called *Oracle* is needed to retrieve information from the side-chain. The Oracle receives monitoring data and inserts them into the transaction network to verify the results.

A. Computation Verification

Verifying the results of computation executed by third-parties is among the biggest challenges in decentralized clouds. The verification needs to prevent malicious actions by providers, who could come up with random (or partially executed) results (in order to save computation time or energy) but still earn the reward for the service, as well as consumers who could claim that results are not correct to harm the reputation of providers. The main computation result verification methods are log verification, correctness checking, and redundant computation. However, logs can

be replaced; correctness checking works only on some classes of problems that could be detected by providers and used to deceive consumers; whereas redundant execution requires extra computation and does not work for stateful applications. For example, several machine learning algorithms are stateful so the whole computation has to be rerun to compare the results. Moreover, results concerned with random numbers or inputs cannot be verified currently. Among existing solutions, [9] requires redundant computing, application code to be open, and ability to stop and resume execution, whereas [10] uses interactive protocols to check proofs that are costly and limited to a class of problems.

Since the verification solutions have several limitations, BDMC platforms use escrow accounts to hold stakes from the involved parties, which are redistributed only between the parties with the verified results. In iExec, for instance, consumers define the trust level, which impacts the level of replication, and stake the funds for compensating the workers that correctly execute the task. In parallel, worker pools advertise their trust level, based on the number of workers in the pool and on the workers' reputation. When chosen, workers also stake funds as a guarantee of correct execution. Workers then send the results to the Proof of Contribution (PoCo) protocol, which evaluates the results based on the majority, reputation and the stakes provided. The scheduler dynamically adds new workers to the task during execution until the required confidence level is reached. All the workers that contributed the correct result are paid an equal share of the consumer's stake; the ones with wrong results have their stakes distributed to the correct ones.

In this work, we improve computation verification in BDMC in three ways. First, we eliminate the assumption in [11] that workers are independent and do not coordinate their behavior. Malicious workers could maintain a high reputation and avoid loss of stakes when they are outnumbered by others in a task, whereas jointly owned workers assigned to the same task could coordinate their outputs to gain the majority and get compensated without processing the task. We demonstrate that this is in almost 5% of the tasks if only 3% of the workers are malicious (see Sec. V-E2). Second, we propose a preemptive approach that determines the set of workers that is most probable to reach a correct consensus before the execution. Gradually adding workers during execution increases response time significantly for computation-intensive tasks in particular. Moreover, the random selection of workers would result in higher redundancy, and hence higher cost, with respect to our dependency-aware mechanism. Finally, we bring inadvertent failures into the equation in contrast to escrow accounts, which are unable to discourage or avoid them.

III. Prediction Models

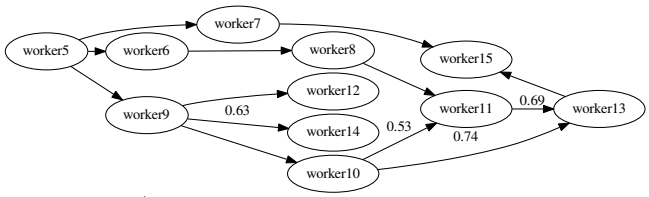
Our hypothesis is that the malicious behavior of different workers exhibits correlation, similar to the inadvertent failures, which are already shown to be correlated in our previous work [5]. The main reason for the correlated malicious behavior is the joint ownership of multiple workers by a malicious entity. We aim to extract the dependencies between workers from historical task logs in order to detect joint ownership. Task success probability can be substantially improved cost-efficiently by avoiding such workers.

Table I
Common Notation and Symbols Used in The Paper

Notation	Definition
m	Number of tasks included in the log traces, $m \in \mathbb{N}$
n	Number of workers present in the platform, $n \in \mathbb{N}$
$r_{i \leq n}$	Random discrete variable representing the status of a worker
r_{n+1}	Random binary variable representing the status of a task
k	Number of replicas (i.e. workers) for a task, $k \in \mathbb{N} \leq n$
t	Trust parameter of a task, $t \in \mathbb{N}$
W	Set of all workers involved in a task, $\|W\| = k$
ω	The output returned by a worker
K	Consensus reached by the workers of a task
G	Structure graph of a Bayesian network, $G = \langle V, E \rangle$
V	Vertices of graph G, $r_i \in V, \|V\| = n + 1$
E	Edges of graph G, $\langle r_i, r_j \rangle \in E$
Θ	Parameters (i.e. CPTs) of a Bayesian network
θ	Conditional probability between variables, $\theta \in \Theta$
D	Set of all log traces, $\|D\| = m$
D_i	Log trace that correspond to a single task, $D_i \in D, r_j \in D_i$
τ	Number of malicious workers in the platform $\tau \in \mathbb{N} \leq n$

Graphical models are commonly used to express the dependency structure between random events. In this work, we utilize BNs, mainly due to their capability to handle conditional dependencies and uncertainty. Another reason is compactness since a BN can specify an exponentially sized probability distribution using a polynomial number of probabilities [12]. BN is a tool to represent and organize the general knowledge about a particular situation. The qualitative part (i.e. structure) of BN is a directed acyclic graph (DAG) with vertices representing relevant variables of the situation and edges representing the dependencies between those. The quantitative part (i.e. parameters), on the other hand, consists of the probabilities of relationships between variables and their parents (direct causes) in the form of Conditional Probability Tables (CPTs) [12]. Consider a set of random variables $R = \{r_1, r_2, \ldots, r_i\}$. A BN can be formally defined for R, as the pair $\langle G, \Theta \rangle$, where G is the above-described DAG, which is a powerful tool also for dependence visualization and independence assumptions. Each variable r_i is directly dependent on its parents in G and independent of its non-descendants given these parents. Then, Θ is a set of conditional probabilities for each variable given its parents. There exists a parameter $\theta \in \Theta$ such that $\theta = \Pr(r_i \mid \text{parents}(r_i))$ for each combination of values that r_i and its parents can possibly take [13]. Computing conditional probabilities of variables that are not directly connected in G are trivial via Bayesian inference.

In our BN model, there exists a variable for each of the n workers $(r_1, r_2, \ldots, r_n)$ indicating its final status (failed,



r_7	r_{13}	$\Pr(r_{15}=0 \mid r_7, r_{13})$	$\Pr(r_{15}=1 \mid r_7, r_{13})$	...
0	0	0.92	0.03	...
0	1	0.88	0.11	...
...		...	...	...

Figure 2. An Excerpt from the BN Structure and the CPT for Worker 15.

correct or incorrect result, etc.) and another binary variable (r_{n+1}) for the final status of the task. We further define the success probability of a task, $\Pr(r_{n+1} = 0)$ in Eq. (1), as the probability that more than half of the workers produce the correct result, namely $\Pr(r_i = 0)$.

$$\Pr(r_{n+1} = 0) = 1 - \Pr\left(\bigcup_{\substack{S \subseteq W \\ |S| = \lceil k/2 \rceil}} \bigcap_{i \in S} r_i \neq 0 \right) \quad (1)$$

Fig. 2 demonstrates an excerpt subgraph from a BN trained with real-world data from the iExec platform. The nodes represent anonymized workers, whereas the links between them are the dependencies inferred from data. For demonstration purposes, we also annotate the links with the corresponding probability values from the CPTs. For instance, $\Pr(r_{14} = 1 \mid r_9 = 1) = 0.63$ is the probability that worker 14 returns an incorrect result given that worker 9 also returns an incorrect result. Thus, greater values indicate stronger dependency between the workers and may also mean that they belong to the same provider. In Fig. 2, we annotate only the probabilities that are greater than 0.5. Indeed, workers 10, 11, and 13 are malicious workers injected into our data set. The dependency between workers 9 and 14, on the other hand, is inferred from the original data set, hence a false positive. Partial CPT in the same figure demonstrates that an error by r_{13} increases the probability of an error by r_{15} from 0.03 to 0.11, given that r_7 is error-free.

Each task is assigned a positive integer trust parameter in the iExec platform. This value indicates the level of trust demanded by the consumer. Here, 1 is the minimum value that corresponds to no trust requirement at all and higher values correspond to increasingly more strict requirements. There is no upper bound for the values that the trust parameter can take. We map the trust parameter to success probability as given in Eq. (2). Note that the required probability is 0 for $t = 1$ and it approaches 1 as t approaches infinity.

$$\Pr(r_{n+1} = 0) \geq 1 - t^{-2} \qquad t \in \mathbb{N} \quad (2)$$

Reasoning in BN models can be performed in both directions. One may predict causes given symptoms, e.g. determining which worker(s) caused the failure, or predict symptoms given causes, e.g. estimating the failure probability given a set of workers [14]. Former is known as diagnostic reasoning, whereas in this study, we focus on the latter that is called predictive reasoning. To this end, we introduce three BN models with variables denoted by $R^A_{i,j}$, where A is model name's initial, i worker, and j task index.

A. Joint Model

In this model, all four possible worker statuses at the end of a task are taken into consideration as distinct cases in a single *joint* BN. Consequently, the first n random variables can take four different values as in Eq. (3).

$$r^J_{i,j} = \begin{cases} 0, & \omega_{i,j} = K_j \\ 1, & \omega_{i,j} \neq K_j \wedge \omega_{i,j} \neq \emptyset \\ 2, & \omega_{i,j} = \emptyset \\ 3, & i \notin W_j \end{cases} \begin{cases} \textit{Worker returns the same result as the consensus.} \\ \textit{Worker returns a different result.} \\ \textit{Worker fails and does not return a result.} \\ \textit{Worker is not involved in the task at all.} \end{cases} \quad (3)$$

The joint model is able to capture the dependencies stemming from possibly malicious intents (case 1) and inadvertent failures (case 2) as well as any interdependencies between those. Thus, it is the most general model.

B. Blend Model

Dependencies stemming from case 1 and case 2 in the joint model are *blended* into a single case 1 in this model. Thus, it does not differentiate between statuses other than the correct result. The cases of the model are given Eq. (4).

$$r^B_i = \begin{cases} 0, & \omega_{i,j} = K_j \\ 1, & \omega_{i,j} \neq K_j \\ 2, & i \notin W_j \end{cases} \begin{cases} \textit{Worker returns the same result as the consensus.} \\ \textit{Worker does not return the same result.} \\ \textit{Worker is not involved in the task at all.} \end{cases} \quad (4)$$

Blending the two cases brings in a more compact search space and CPT tables, hence less inference overhead. However, it may also result in the loss of information that is vital to the accurate prediction of task success.

C. Dual Model

In the dual model, the two causes of dependency are captured in separate BN models. The first model, given in Eq. (5), considers only the failure-free completion disregarding the correctness of the results.

$$r^F_i = \begin{cases} 0, & \omega_{i,j} \neq \emptyset \\ 1, & \omega_{i,j} = \emptyset \\ 2, & i \notin W_j \end{cases} \begin{cases} \textit{Worker returns a result.} \\ \textit{Worker fails and does not return a result.} \\ \textit{Worker is not involved in the task at all.} \end{cases} \quad (5)$$

The second, on the other hand, equate non-involvement and failure, only differentiating between correct and incorrect results. Thus, it focuses only on malicious intent.

$$r_i^M = \begin{cases} 0, & \omega_{i,j} = K_j \\ 1, & \omega_{i,j} \neq K_j \wedge \omega_{i,j} \neq \emptyset \\ 2, & \omega_{i,j} = \emptyset \vee i \notin W_j \end{cases} \begin{cases} \textit{Worker returns the same result as the consensus.} \\ \textit{Worker returns a different result.} \\ \textit{Worker does not return a result.} \end{cases} \quad (6)$$

As a consequence of having two models, there are two different conditional probabilities between all pairs of variables in the dual model. The final probabilities are computed through the intersection of the two models as shown in Eq. (7). This model assumes that failure and malicious events are independent (i.e. $\Pr(r_i^m \mid r_i^f) = \Pr(r_i^m)$ and vice versa).

$$\begin{aligned} \Pr(r_i^D = 0) &= \Pr(r_i^F = 0 \cap r_i^M = 0) \\ &= \Pr(r_i^F = 0) \Pr(r_i^M = 0 \mid r_i^F = 0) \\ &= \Pr(r_i^F = 0) \Pr(r_i^M = 0) \end{aligned} \quad (7)$$

The dual model comes with an additional overhead due to the presence of two BNs. It is not able to detect the dependencies between these two, either. However, it might model the dependencies of the same type in greater accuracy in the absence of noise due to the other type's interference.

IV. Reliability-aware Task Replication

Distributed clouds are mainly intended for time-sensitive services that are negatively affected by the delay overhead of the reactive reliability techniques such as checkpointing and re-execution. Thus, in practice, proactive replication is usually implemented [15]. In this section, we present our second major contribution that is a task scheduling algorithm for estimating the optimum set of workers for a task in terms of its success probability. The main goal of the **R**eliability-**a**ware **Ta**sk **R**eplication (`RaTaR`) algorithm is to probabilistically guarantee that the selected workers, where the task is replicated, reach the correct consensus in bounded time. A consensus cannot be reached if either too many workers fail or there is no majority in terms of outputs. Worse still is the case that an incorrect output constitutes the majority. As discussed earlier, this may be the case when several workers are owned by the same malicious entity that aims to get compensated without work or simply to harm. Given a cost budget of the number of replicas (k), `RaTaR` identifies the set of workers with the least inter-dependency, and hence the lowest probability of joint ownership.

An exhaustive evaluation of all candidate sets would result in $\mathcal{O}(C(n,k))$ or $\mathcal{O}(n^k)$ time-complexity. Instead, we approximate the optimum solution by selecting only one node from each dense graph partition. To that end, we first partition the graph into k subgraphs and then evaluate all combinations of subsets that contain exactly one element from each subgraph V_i. Although optimum minimum-cut graph partitioning itself is NP-hard, state-of-the-art heuristics can reach approximate solutions in near-linear time [16]. In the worst case, subgraphs are of equal size and search complexity is $\mathcal{O}(n^k/k^k)$ as shown in Eq. 8.

$$\prod_{i=1}^{k} \binom{|V_i|}{1} = \prod_{i=1}^{k} |V_i| \approx \prod_{i=1}^{k} \frac{n}{k} = \frac{n^k}{k^k} \quad (8)$$

An example near-balanced partition set with $k = 3$ for the graph in Fig. 2 can be $P_3 = \{\{9, 12, 14\}, \{8, 10, 11, 13\}, \{5, 6, 7, 15\}\}$. This partitioning ensures that only one of the

Algorithm 1 Reliability-aware Task Replication

Input BN structure graph: $G = \langle V, E \rangle$
The number of replicas: $k \in \mathbb{N}$
Trust needed by the task: $t \in \mathbb{N}$
Output The set of selected workers for replication: R

$G' \leftarrow \langle V', E', W \rangle$ # Define a weighted graph G'
$V' \leftarrow V, E' \leftarrow \emptyset, W \leftarrow \emptyset$ # G' is an empty graph
for all $\langle r_i, r_j \rangle \in E$ **do** # For each dir. edge in G
  $E' \leftarrow E' \cup \{r_i, r_j\}$ # Add an undir. edge to G'
  $W \leftarrow W \cup \langle \{r_i, r_j\}, \Pr(r_j \mid r_i) \rangle$ # Add weight...
end for # ...to the edge based on BN
$\mathrm{P}[1 \ldots k] \leftarrow \mathrm{partition}(G', k)$ # Partition the graph
$\mathrm{I}[1 \ldots k] \leftarrow 0$ # I holds the partition indices
$p_{max} \leftarrow 0$ # Maximum success probability
while true **do**
  $C \leftarrow \emptyset$ # C is an initially empty combination
  for $i = 1$ to k **do** # Add the next element...
    $C \leftarrow C \cup \mathrm{P}[i][\mathrm{I}[i]]$ # ...from each partition
  end for
  $p \leftarrow \Pr(r_{n+1} = 0)$ # Success probability (1)
  if $p > p_{max}$ **then** # C is currently the...
    $p_{max} \leftarrow p$ # ...best known solution
    $R \leftarrow C$ # Update the output value
  end if
  if $p \geq 1 - 1/t^2$ **then** # If t is satisfied...
    break # ...then stop the search
  end if
  $r \leftarrow k$ # The rightmost unexhausted partition
  while $r > 0$ **and** $\mathrm{I}[r] + 1 > |\mathrm{P}[r]|$ **do**
    $r \leftarrow r - 1$ # Compute r by iterating over P
  end while
  if $r < 1$ **then** # If no combination is left...
    break # ...then stop the search
  end if
  $\mathrm{I}[r] \leftarrow \mathrm{I}[r] + 1$ # Move to the next element of r
  for $i = r + 1$ to k **do** # Reset the partitions...
    $\mathrm{I}[i] \leftarrow 0$ # ...to the right of r
  end for
end while

Figure 3. Pseudo-code description of the `RaTaR` algorithm.

workers 10, 11, and 13 can be chosen for a task. Moreover, the number of subsets to be evaluated is reduced from 165 (i.e. $C(11,3)$) to only 48 (i.e. $3 \cdot 4 \cdot 4$).

Fig. 3 presents the pseudo-code description of the `RaTaR` algorithm. The first step (lines 1–6) is the conversation of the BN structure G, which is a directed graph, into an undirected one (G') and its annotation with edge weights of conditional probabilities. Therefore, G' can be clustered into k partitions in line 7. The output P is a two-dimensional array where $\mathrm{P}[i][j]$ contains the node j in partition i. The rest of the algorithm iterates over all combinations of k partitions picking exactly one node from each partition by maintaining an index for each partition I (line 8), adding the indexed element from each partition to the set C (12–14), and updating the indices (23–26, 30–33). For each combination C, it computes the success probability p and detects the combination R with the highest p (15–19). The search stops prematurely if the required trust level (t) is reached (20–22) or normally if all combinations are exhausted (27–29).

V. Evaluation

A. Data Set

The experimental data set is composed of traces from over 3,000 tasks that ran on the iExec platform in June 2019. iExec's public cloud computing marketplace is composed of multiple worker pools, each made up of one scheduler and of multiple untrusted workers, as described in Sec. II. Each record in the data set contains metadata about a task from the main public worker pool, indicating which application (Docker container) was run, selected trust level, address of the workers that contributed to the task, timestamped status, and consensus result. Worker addresses link to their on-chain reputation, which can be queried separately.

B. Baselines

1) Naive Bayes (NB): NB is a generative model similar to the BN. Yet, it is much simpler than BN as it assumes that all attributes (workers) are conditionally independent of each other given the class value (failure). Improvement of BN over NB would quantify the significance of the dependencies between the workers to the failure prediction.

2) Random Forest (RF): As an alternative to generative models, we evaluate RF, which is a discriminative model. It trains multiple decision trees and makes ensemble decisions. It is known to be robust against dimensionality and unbalanced classes [17], which are present in our use case.

3) Logistic Regression (LR): LR, also a discriminative model, is one of the simplest but most widely used machine learning algorithms. Different from RF, it assumes that there exists a linear relationship between the attributes. When this assumption does not hold it could be outperformed by RF.

4) No Prediction (NO): This baseline represents the worst case that the failures are not predicted at all. In other words, all sets of workers are expected to execute fault-free.

Table II
Experimental Parameters

Parameter		Definition	Value
MC	r	Variable values	*See Sec. V-D*
	k	Number of replicas	Lognormal$(0.4, 0.6)$
Static	m	Number of tasks	3172
	n	Number of workers	328
		Number of iterations	100
		Number of CV folds	10
BN	$-P$	Maximum number of parents	3
	$-A$	Initial count for BMA	0.5
RF	$-P$	Size of each bag	100%
	$-I$	Number of iterations	1000
LR	$-R$	Ridge value in the log-likelihood	10^{-8}

C. Simulation Environment

BDMC environment is multidimensional in terms of parameters. Moreover, runtime performance is highly sensitive to the values that parameters take. Thus, the outputs of the performance evaluations vary greatly. To overcome this difficulty and to obtain results with high confidence, we resort to Monte Carlo simulation, which is successfully applied to distributed computing environments [18], [19]. The main idea is to replace any parameter that has inherent uncertainty with a probability distribution and repeat the simulation with as many random values of these parameters as needed to reach the intended confidence interval.

The simulation environment is developed in Java and executed on an Intel Xeon E5-2650 processor with 256 GB random access memory. Weka 3 [20] implementation of the machine learning algorithms is used via Java API of Weka. These correspond to Cooper and Herskovits's hill-climbing algorithm for BN training [21], John and Langley's NB [22], Le Cessie and Van Houwelingen's LR [23], and Breiman's RF classifier [24]. Table II list the parameters used in the experiments. The first two are the Monte Carlo parameters with random values. r indicates the final status (fail, incorrect result, correct result, etc.) of each worker at the end of each task as described in Sec. V-D. k, on the other hand, is the number of workers that participate in each task. For this, we used the real values in the data set, which fits a Lognormal distribution. For each algorithm, we run a sensitivity analysis to identify the parameter values with the highest accuracy (NB is parameter-free). Each configuration is evaluated 1000 times using 10-fold cross-validation.

D. Threat Injection

Since the failures in real-world cloud systems are relatively rare, it is not feasible to collect enough training data so that the dependencies can be accurately captured. Moreover, there currently does not exist a malicious provider detection policy on the platform. Thus, it is not possible to validate that detected dependencies by the BN are accurate. Instead, we inject malicious workers into the data set to evaluate detection accuracy. This process is described via pseudo-code in Fig. 4. Here, we first randomly select τ workers to

Algorithm 2 Injection of Malicious Workers and Threats

Input The set of data instances: $D = \{D_1, D_2, \ldots, D_m\}$
The number of threats to be injected: $\tau \in \mathbb{N}$

Output The updated set of data instances: D

1: $W \leftarrow \{1, 2, \ldots, n\}$ # The set of worker indices
2: $T \leftarrow \text{randomSample}(W, \tau)$ # τ malicious workers
3: **for all** $D_i \in D$ **do** # For each data instance in D
4: $a \leftarrow 0$ # The number of malicious workers
5: **for all** $r_j \in D_i$ **do** # For each variable in D_i
6: **if** $j \in T$ **then** # If r_j is malicious, change...
7: $r_j \leftarrow x$ # ...status to a placeholder
8: $a \leftarrow a + 1$
9: **end if**
10: **end for**
11: **for all** $r_j \in D_i$ **do** # For each variable in D_i...
12: **if** $r_j == x$ **and** $a \geq 2$ **then** # If multiple...
13: $r_j \leftarrow 1$ # ...change output to incorrect
14: **else**
15: $r_j \leftarrow 0$ # Otherwise, keep it correct
16: **end if**
17: **if** $a \geq \max\left(\frac{|D_i|}{2}, 2\right)$ **then** # If majority...
18: $r_{n+1} \leftarrow 1$ # ...change status to failed
19: **end if**
20: **end for**
21: **end for**

Figure 4. Pseudo-code description of the threat injection algorithm.

be considered malicious (lines 1–2) and then iterate over the whole data set to inject malicious behavior (3–21) by these workers. The final status of a task changes only if involved malicious workers form the majority (17–19).

Our threat injection mechanism also mimics the prudent behavior of malicious workers to avoid detection. They produce incomplete or incorrect results only when they have the majority of the workers on a particular task so that their results would be the consensus and they will be compensated. However, the iExec platform does not allow workers to detect which or how many other workers are assigned to the same task as them. Thus, they can only predict having the majority. In the iExec data that we use in this work, the mean number of workers assigned to a task is 2.44. Accordingly, injected threats act maliciously when at least two of them are assigned to a task (lines 12–16) as this would suffice to have the majority with a high probability.

E. Numerical Results

1) Failure Prediction: The first step in reliable worker selection is an accurate prediction of whether a particular set of workers would result in a task failure or not. Detecting most of the failures (high recall) is our main priority. False positives, that are non-failing sets classified as failing (low precision) is tolerable to some extent. Since failures are usually rare, low precision should not have a strong impact on overall accuracy. We present overall accuracy and recall scores of all algorithms in Figs. 5 and 6.

We observe a strong decay in accuracy with the baselines NO and NB as the number of injected malicious workers increases. This is because NO does not predict the failures at all and NB assumes that the behavior of the workers is conditionally independent. It is clear from the results that the independence assumption is not valid in practice and more advanced machine learning algorithms that capture said dependencies are imperative. Accuracy of the BN, RF, and LR algorithms are comparable, but LR is able to maintain a slightly higher accuracy for a greater number of malicious workers. However, when we turn to recall results, LR is notably the worst performer. This indicates that LR tends to classify tasks as success, which results in low recall but high accuracy due to the rarity of failure cases. Considering the importance of recall in our case, LR does not turn out to be a suitable algorithm for failure prediction. On the contrary, NB exhibits relatively high recall, despite its low overall accuracy. The best recall scores are achieved by the proposed BN approach, closely followed by RF in high threat cases (i.e. $\tau \geq 20$). the recall score of NO baseline is always zero since there is no learning. Other algorithms generally achieve higher recall as τ is incremented due to the increasing availability of failure data to train the models.

BN and RF outperform other algorithms in accuracy and recall due to their capability of utilizing worker dependencies in prediction. We present more detailed results in Figs. 8 and 9 in order to determine the winner between these two and the confidence of the results. In accuracy scores, BN is slightly better as it outperforms RF in six values of τ in comparison to only one value that it is worse. In four cases, the accuracy scores of two algorithms lie within the 99.9% confidence interval (CI). In terms of recall, on the other hand, BN is clearly ahead with a higher score in 10 of 11 τ values. RF particularly underperforms at low τ values, which indicates that it fails to learn from limited data. Another observation from Fig. 9 is the significantly wider CI area of RF in comparison to BN. Recall values achieved by RF vary greatly with the random selection of workers, whereas BN is much more robust to randomness.

2) Failure Avoidance: In the second part of our evaluation, we measure the actual improvement of the proposed prediction models on failure avoidance using the `RaTaR` algorithm and BN. To this end, we report the number of failures occurred with each model based on real task logs. These are the consequence of sets of workers that are erroneously evaluated as reliable by the models but failed their tasks. Fig. 7 compares the prediction models and `RaTaR` algorithm to the default selection of workers in the iExec platform. We observe that if only 3% of the workers are malicious, 144 tasks ($\approx 5\%$) are affected on average. This increases to almost 704 failures ($\approx 22\%$) with the 30% of the workers. Default algorithm results in at least 9.8

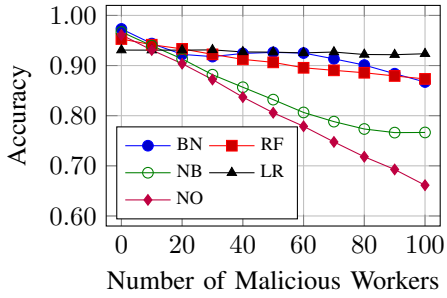


Figure 5. Accuracy Scores of Baselines.

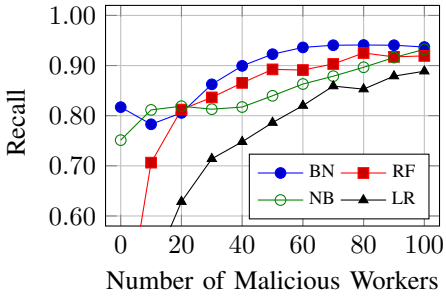


Figure 6. Recall Scores of Baselines.

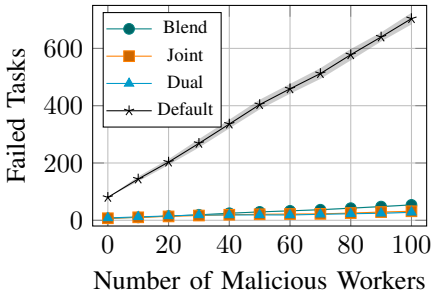


Figure 7. Failure Avoidance of RaTaR Models.

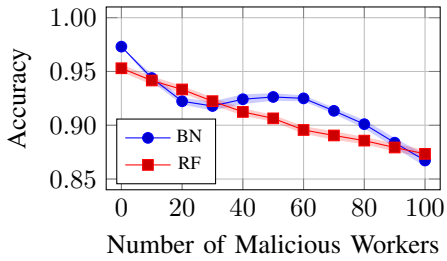


Figure 8. Detailed Accuracy Scores.

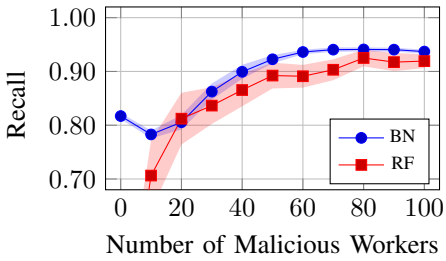


Figure 9. Detailed Recall Scores.

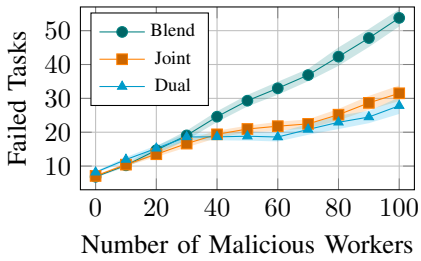


Figure 10. Detailed Failure Avoidance of RaTaR.

times (Dual model, $\tau = 0$) and up to 26 times (Dual model, $\tau = 90$) more failed tasks than the proposed algorithms. The results clearly demonstrate that BN models and RaTaR are highly effective in scheduling tasks to reliable workers.

We also compare the three prediction models in Fig. 10. Although BN models predict better as τ increases (as shown by previous results), more and more failures still occur due to the greater number of malicious workers. The models perform comparably close until $\tau = 30$, however, they start to diverge after this point. Between the joint and dual models, the latter is slightly better, although 99.9% CI of the two often intersects. As a consequence of ignoring the distinction between malicious actions and failures, the blend model is less accurate than the former two.

3) Runtime Performance: Fig. 11 presents the BN training times for the three prediction models. The absence of the fourth value for random variables in the blend model does not result in noticeable runtime performance improvement to joint model. The dual model, on the other hand, take almost double time to train due to the presence of two separate BNs. We believe, barely noticeable improvement of prediction performance in the dual model (in comparison to the joint) does not justify its exorbitant training overhead.

Furthermore, we analyze the sample complexity of the algorithms that is the number of training samples required to converge to their optimal performance. In the previous experiments, we use 90% of the data for training as a result of 10-fold cross-validation. Here, we start with a tiny data set of 100 instances (≈3%) and gradually increase its size. Fig. 12 shows the recall scores of the algorithms normalized to the $[0, 100]$ range. We observe that all algorithms reach their peak performance with at most 2,000 instances. BN and NB have the lowest sample complexity and are able to show their optimal performance with 1,200 data instances. RF also reaches 97% around the same point but it peaks only at 2,000. Moreover, its early performance (up to 700) is significantly lower than BN and NB, whereas LR converges even slower. These results explain the low performance of RF and LR with small τ values in Fig. 6.

4) Cost Reduction: It is of interest to evaluate whether proposed algorithms can maintain the reliability with fewer replicas, hence the lower cost. The deciding factor here is the trust parameter, which is agreed between the provider and consumer in the Service Level Agreement [25]. In our data set, the vast majority of tasks (3,114 out of 3,172) require the trust levels of either one or four. Thus, we limit our evaluation in Fig. 13 to these two values. As expected, fewer failures occur with a high trust level due to the greater number of workers involved in these tasks. The percentage of failed tasks declines from 22% to 20% with the default algorithm and from 0.71% to zero with RaTaR. We also implement two alternative algorithms with a fixed number of workers assigned to each task. We would like to remind that the average number of replicas is 2.44 in our data set. With two replicas (Joint–2), RaTaR is able to maintain almost the same failure rate when $t = 1$, but increases to 0.18% when $t = 4$. It can also eliminate almost all failures when three replicas (Joint–3) are feasible. These show that RaTaR can outperform the default algorithm even with fewer workers and emphasize the need for a dynamic number of workers.

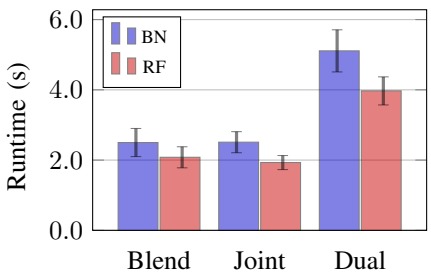


Figure 11. Runtime Performance Comparison.

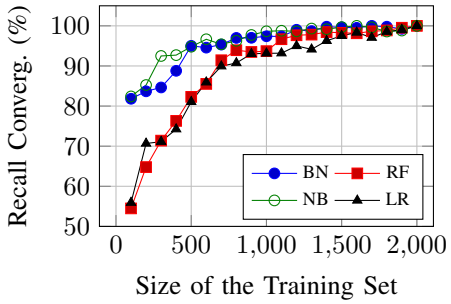


Figure 12. Sample Complexity and Convergence.

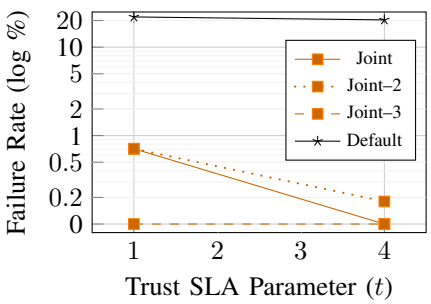


Figure 13. Impact of Replica Count and Trust.

VI. Related Work

Failure resilience in the decentralized cloud is still an open issue [26] despite its necessity for wide adoption. An early discussion of reliability challenges in fog computing is presented in [27]. Aral and Brandic [5], [28] introduce a technique that exploits causal relationships between different types of failures and channel all QoS related parameters through availability. Nebula [29], a fog based computation and storage architecture, ensures fault tolerance of compute nodes through re-execution. Although data is replicated, reliability is not a factor in replica site selection. Cloud visitation platform [30], which copes with hardware heterogeneity problem in multi-clouds via hardware awareness, solves failure resilience only at a local level. That is, deployed applications are migrated to another node, after a server failure. Li et al. [31] discuss a scheme for cloud storage based on blockchain. Even though reliability and redundancy are in the scope of the work, the dependency between workers is not analyzed to improve storage allocation. Probabilistic model checking is another technique that is recently shown to be effective for failure avoidance [32].

There exist a plethora of scientific works on the problems of contribution validation and reputation management [33]. However, given the recent introduction of BDMC and the new challenges this brings, existing mechanisms for grid, peer-to-peer, or similar architectures, require modifications. For instance, Sarmenta proposes a sabotage-tolerance mechanism based on credibility scores for volunteer computing systems [11], however, in such systems financial incentives of the workers are not relevant. Additionally, the attacks are assumed to be independent in this work. More recently, a blockchain-based authentication scheme for mobile clouds [34] is proposed. Here, mobile devices participating in the mobile resource pool are prevented to falsify data.

VII. Discussion and Conclusion

Smart Contracts and blockchain have the potential to reshape the cloud market and boost the adoption of clouds and fog computing. However, since BDMC enables small providers to monetize resources and they are particularly prone to failures, these aims can only be achieved through increased reliability of cloud services and providers. This paper proposes a model to predict the probability of failure-free execution and correct computational task outputs by relying on the potential dependencies between providers extracted from historical log traces. This model is highly effective in detecting malicious providers that secretly own multiple workers with the intent of getting incomplete results through while still getting compensated. We also propose a probabilistic scheduler, `RaTaR`, based on the prediction model and evaluate it using real runtime traces from the iExec decentralized cloud platform.

The results demonstrate that dependency-aware models (e.g. Bayesian networks) are able to identify reliable workers with significantly higher accuracy than others. Moreover, tasks scheduled through `RaTaR` result in up to 96% fewer failures or incorrect outputs even though they utilize 18% fewer workers on average. Thus, the proposed technique can not only increase reliability but also decrease resource costs. The reliability model presented here can be used in different contexts and as part of other methodologies, in particular for the composition of provider reputation and for reducing failures in service execution. Moreover, although the `RaTaR` scheduler is devised mainly to validate the reliability model and it could be easily extended to cover other crucial aspects of scheduling services, such as response time, availability and hardware characteristics.

Predicting the reliability of cloud providers is only useful in this context if malicious or providers with bad reputation are penalized and prevented to create new profiles, as historical information would be lost and the new profiles would be neutral. However, before penalizing new profiles, we need to consider one of the main benefits of BDMC: the very capacity to facilitate the entry of new providers. Therefore, new solutions to find the right balance between these conflicting needs to be developed. Furthermore, our prediction model relies on logs and historical data and its precision increases with more data. Therefore, it requires a certain amount of data before its deployment. Moreover, it needs periodical retraining. We believe, however, that

devising an online machine learning methodology for the prediction model as future work could enable it to cope better with changes in the dependencies. Finally, we leave the decision on the number of workers assigned to each task as future work. `RaTaR` can be extended to optimize the replication cost based on the trust parameter from the consumer and conditional probabilities from the BN model. Our initial results with different trust parameters and the fixed number of replicas are promising in this direction.

ACKNOWLEDGMENT

This work has been partially funded through the Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015 and EU H2020 Marie Sklodowska-Curie Grant Agreement No.838949. The authors would like to thank Onat Kaya for his support in log parsing.

REFERENCES

[1] R. B. Uriarte and R. De Nicola, "Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards," *IEEE Com. Stand.*, vol. 2, no. 3, pp. 22–28, 2018.

[2] Z. Shi *et al.*, "Operating permissioned blockchain in clouds: A performance study," in *ISPDC*, 2019, pp. 50–57.

[3] R. B. Uriarte *et al.*, "Towards distributed SLA management with smart contracts and blockchain," in *IEEE CloudCom*, 2018, pp. 266–271.

[4] R. Buyya *et al.*, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *ICA3PP*, 2010, pp. 13–31.

[5] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *IEEE/ACM SEC*, 2018, p. 228.

[6] G. Fedak *et al.*, "Blockchain-based decentralized cloud computing," iExec Blockchain Tech, Tech. Rep., 2018.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," bitcoin.org, Tech. Rep., 2008.

[8] X. Xu *et al.*, "A taxonomy of blockchain-based systems for architecture design," in *IEEE ICSA*, 2017, pp. 243–252.

[9] R. Canetti *et al.*, "Practical delegation of computation using multiple servers," in *ACM CCS*, 2011, pp. 445–454.

[10] S. Setty *et al.*, "Making argument systems for outsourced computation practical (sometimes)," in *NDSS*, 2012, p. 17.

[11] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for vol. comp. systems," in *IEEE/ACM CCGrid*, 2001, pp. 337–346.

[12] A. Darwiche, *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[13] N. Friedman *et al.*, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[14] K. B. Korb and A. E. Nicholson, *Bayesian artificial intelligence*. CRC press, 2010.

[15] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE TNSM*, vol. 15, no. 2, pp. 516–529, 2018.

[16] D. A. Spielman and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning," *SICOMP*, vol. 42, no. 1, pp. 1–26, 2013.

[17] R. B. Uriarte *et al.*, "Supporting autonomic management of clouds: Service clustering with random forest," *IEEE TNSM*, vol. 13, no. 3, pp. 595–607, Sep. 2016.

[18] A. Aral and V. De Maio, "Simulators and emulators for edge computing," in *Edge Computing: Models, Technologies and Applications*, J. Taheri and S. Deng, Eds. IET, (in Press).

[19] V. De Maio and I. Brandic, "First hop mobile offloading of DAG computations," in *IEEE/ACM CCGrid*, 2018, pp. 83–92.

[20] I. H. Witten *et al.*, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[21] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine learning*, vol. 9, no. 4, pp. 309–347, 1992.

[22] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *UAI*, 1995, pp. 338–345.

[23] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Journal of the Royal Statistical Society: Series C*, vol. 41, no. 1, pp. 191–201, 1992.

[24] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[25] R. B. Uriarte *et al.*, "Defining and guaranteeing dynamic service levels in clouds," *FGCS*, vol. 99, pp. 27 – 40, 2019.

[26] R. Roman *et al.*, "Mobile edge computing, fog et al." *FGCS*, vol. 78, pp. 680–698, 2018.

[27] H. Madsen *et al.*, "Reliability in the utility computing era: Towards reliable fog comp." in *IWSSIP*, 2013, pp. 43–46.

[28] A. Aral and I. Brandic, "Quality of Service Channelling for Latency Sensitive Edge Applications," in *IEEE EDGE*, 2017, pp. 166–173.

[29] M. Ryden *et al.*, "Nebula: Distributed edge cloud for data intensive computing," in *IEEE IC2E*, 2014, pp. 57–66.

[30] M. Zhanikeev, "A cloud visitation platform to facilitate cloud fed. and fog comp." *Computer*, vol. 48, pp. 80–83, 2015.

[31] J. Li *et al.*, "Block-secure: Blockchain based scheme for secure p2p cloud storage," *Info. Sci.*, vol. 465, p. 219, 2018.

[32] J. Zilic, A. Aral, and I. Brandic, "EFPO: Energy efficient and failure predictive edge offloading," in *IEEE/ACM UCC*, 2019, pp. 165–175.

[33] A. E. Arenas *et al.*, "Reputation management in collaborative computing systems," *SCN*, vol. 3, no. 6, pp. 546–564, 2010.

[34] H.-W. Kim and Y.-S. Jeong, "Secure authentication-management human-cent. scheme for trusting personal resource info. on MCC with blockchain," *HCIS*, vol. 8, 2018.