BlockAM: An Adaptive Middleware for Intelligent Data Storage Selection for Internet of Things

***Abstract*—Current Internet of Things (IoT) infrastructures, with its massive data requirements, rely on cloud storage: however, usage of a single cloud storage can place limitations on the IoT applications in terms of service requirements (performance, availability, security etc.). Multi-cloud storage architecture has emerged as a promising infrastructure to solve this problem, but this approach has limited impact due to the lack of differentiation between competing cloud solutions. Multiple decentralized storage solutions (e.g., based on blockchains) are entering the market with distinct characteristics in terms of architecture, performance, security and availability and at a lower price compared to cloud storage. In this work, we introduce BlockAM: an adaptive middleware for the intelligent selection of storage technology for IoT applications, which jointly considers the cloud, multi-cloud and decentralized storage technologies to store large-scale IoT data. We model the cost-minimization storage selection problem and propose two heuristic algorithms: Dynamic Programming (DP) based algorithm and Greedy Style (GS) algorithm, for optimizing the choice of data storage based on IoT application service requirements. We also employ blockchain technologies to store IoT data on-chain in order to provide data integrity, auditability, and accountability to the middleware architecture. Comparisons among the heuristic algorithms are conducted through extensive experiments, which demonstrate that the DP heuristic and GS heuristic achieve up to 92% and 80% accuracy respectively. Moreover, the price associated with a specific IoT application data storage decrease by up to 31.2% by employing our middleware solution.**

I. Introduction

The dramatic deployment of IoT applications worldwide results in the production of terabytes of heterogeneous data per day, ranging from a few bytes of device statuses, sensor readings, to large video/audio files. Since IoT devices possess limited computational power, memory, and/or energy availability, there is a need to store and manage the data efficiently. Cloud storage has emerged as a scalable and promising infrastructure to store and manage large-scale IoT data. Normally, IoT devices generates raw data which is preprocessed before being stored. A fog/edge server, near to IoT devices, performs aggregation and filtering of raw IoT data. Fog nodes extract the time-critical IoT data and forward the archived data to cloud premises for future storage and retrieval as shown in Fig. 1 [1]. Unfortunately, current centralized cloud storage solutions bring significant security and trust issues as sensitive data from the clients can easily be leaked from the servers [2]. Moreover, cloud providers can intentionally hide the data errors for their own benefits, which directly affects the data integrity [3].

Once the data is stored on the cloud or other storage technologies, it is physically not accessible to the IoT application users. There is a need to assure the IoT application user of its data integrity [4]. Therefore, it is necessary for the end users to efficiently perform the data integrity verification without storing the local copy of data files. Moreover, auditability and accountability of the data storage decisions, data access records and data provenance is important to ensure trust and confidence, while dealing with untrusted storage technologies [5]. Thus, there is a need to introduce auditability and accountability mechanisms in untrusted environment to provide security and trust, while mitigating multiple security threats along with the reduction in risks and failures.

Recently, decentralized storage solutions are gaining attraction as an alternative to cloud storage. Such solutions are usually based on a combination of blockchain technology [6] and peer-to-peer (P2P) networking [7]. The main aim of these decentralized storage solutions is to give the clients more control over their data and to provide security without requiring trusted third parties. Sia [8], Storj [9], Filecoin [10], Safecoin [11] are few of the emerging decentralized storage solutions. Blockchain solution can be leveraged to provide data integrity, auditability and accountability to IoT application. Once the data is stored in the blockchain network, it cannot be deleted or modified therefore, by storing the data hash and the data management decisions e.g. the data provenance information and data migration decision for the IoT applications in the blockchain network, data integrity, auditability and accountability can be ensured [12].

IoT applications have different service requirements in terms of performance, security, privacy, availability and price [13]. However, relying on a single cloud storage puts limits on the applications, which could be alleviated by another storage solution [14]. Multi-cloud storage architecture [15], [16], [17], [18], [19] has recently been proposed to solve this problem, but this approach has limited impact due to the lack of differentiation between competing cloud solutions. On the other hand, decentralized storage solutions have distinct characteristics in terms of architecture, performance, security and availability as compared to cloud storage. For instance, unlike cloud storage, the data in decentralized storage technologies is split and stored in multiple storage locations around the world, thus preventing intentional/unintentional data leakages. Similarly, in contrast to cloud storage, performance of these decentralized storage systems directly depends on the type of underlying blockchain network and consensus protocols. Moreover, these decentralized solutions are lower priced compared to cloud [9], [10].

Therefore, a certain decentralized storage solution may be more effective than cloud storage depending on the IoT application data storage pattern. Hence, the data placement decisions should be based on service requirements and parameters for individual IoT applications. For instance, the IoT healthcare applications require strict service requirement of privacy compared to smart home temperature sensor IoT application. Similarly, CCTV camera-based IoT applications generate a large amount of data compared to a simple sensing IoT application. In addition, price reduction is the core objective of almost every IoT application, which motivates the consideration of multiple storage technologies to select the one with least price. Furthermore, the characteristics of these decentralized storage technologies are subject to dynamicity (e.g. performance variation, changing price etc.) and differ significantly from traditional cloud. These service requirements and dynamic properties of IoT applications as well as storage technologies need to be monitored to enable in-depth visibility of the real-time parameters. Moreover, there is a need to store up-to-date service requirements of individual IoT applications to decide the best-suited storage technology. These two points necessitate a middleware design that takes into consideration the real-time parameters of storage technologies as well as IoT applications along with up-to-date service requirements in order to perform the data placement decision.

Existing works address the problem of large scale IoT data storage by employing the blockchain-based solutions [20], [2], [21], [22]. However, the problem of considering multiple storage solutions simultaneously with auditable data provenance has been largely ignored. Moreover, authors in [14], [23] proposed the adaptive middleware which decides the storage solution based on the requirement of applications however, the work only considers the cloud storage technologies. In other words, the proposed middleware doesn't consider the decentralized storage technologies. Furthermore, it lacks the mathematical formulation of multi-objective decision optimization problem and ignores the compelling properties of the middleware e.g traceability, auditability and accountability. The above mentioned problems are the main focus of our work.

In this work, we introduce BlockAM: an adaptive middleware-based IoT fog architecture with blockchain network for intelligent selection of storage technology, given an individual IoT application service requirements and to integrate different kinds of decentralized storage technologies with cloud and multi-cloud architecture. The adaptive middleware considers the dynamic behavior of storage solutions as well as IoT application properties to select the best combination of storage solutions, while fulfilling the specified service requirements for the IoT applications. This paper is a study of this theoretical problem, its proposed practical solutions along with the evaluation of said solutions.

Our proposed middleware:

1) performs continuous monitoring of the storage solutions as well as IoT application parameters in real time and keeps an up-to-date record of them,
2) maintains service requirements for individual IoT application and use this information to decide the best-suited storage,
3) decides the optimal storage solution for specific IoT application's service requirement by taking into consideration different parameters i.e. performance, security, privacy, availability, price etc.
4) records the storage decision and hash of the data to a blockchain network to ensure data integrity, traceability and accountability properties,
5) manages the migration of data from one storage solution to another and make predictions about future migrations based on past parameters.

The middleware performs multi-objective optimization of the cost function represented by different storage solution characteristics, subject to multiple application service requirements parameters. Our contributions in this work are:

1) We propose an extended fog architecture for IoT application with blockchain network, which takes into consideration the decentralized storage technologies along with the traditional cloud storage (Section III-B).
2) We provide a survey of different decentralized storage technologies and extensive comparison in terms of security, privacy, performance etc (Section IV).
3) We formulate the data storage selection problem as a multi-objective decision optimization problem that finds the storage solution based on the captured cost functions of storage technologies and service requirements of IoT applications. This optimization problem can easily be extended for different storage technologies as well as service requirements and is proven to be NP-Hard (Section V-A).
4) We propose two dynamic programming and greedy-style based polynomial-time heuristic algorithms to approximate a solution to the multi-objective optimization problem (Section VI).
5) We design and implement each component of our adaptive middleware in Python (Section VII).
6) We experimentally evaluate our middleware design along with the performance of proposed heuristic algorithms with baseline. Our results show that our middleware effectively reduces the price of IoT data storage while maintaining service requirements (Section VIII).

The rest of the paper is organized as follows. Section II presents the related works for the paper. Section III describes the traditional and proposed adaptive middleware IoT fog architecture. Section IV presents the survey of decentralized storage technologies. Section V describes the theoretical problem formulation of the middleware design. Section VI describes the two proposed heuristic algorithms. Section VII and Section VIII presents the implementation details and main evaluation results of the proposed solution respectively. Finally, Section IX summarizes the key conclusions of our paper.

II. Related Works

In this section, we review the related studies on the blockchain-based IoT data storage [20], [2], [21], [22], multi-cloud storage systems [15], [16], [17], [18], [19], [23], [14], decentralized storage technologies [8], [9], [10], [11], and draw a comparison between these studies and our work.

Decentralized storage solutions have recently gained attention as an alternative to current cloud storage. These decentralized storage technologies are usually based on a combination of blockchain technology and peer-to-peer (P2P) networking. One may think of these decentralized storage technologies as the "Airbnb of storage", where storage providers generate revenue by renting their unused drive space [7]. Sia [8], Storj [9], Filecoin [10], Safecoin [11] are few of the emerging decentralized storage solutions. Storj uses extended proof-of-retrievability i.e. probabilistic audit spot checks to make sure storage nodes are storing the data for defined time [9]. Filecoin utilizes Proof-of-Replication to make sure the storage node has replicated the data to its dedicated physical storage, while to prove that data is being stored for a defined period of time, Filecoin employs Proof-of-Space-Time, where a prover node generates the storage-proofs recursively in the blockchain network [10].

In recent years, blockchain technology has emerged to solve security and privacy issues in large scale IoT data storage. In [20], authors proposed a blockchain-based storage mechanism for time series IoT data in which they employed Distributed Hash Table (DHT) as an off-chain storage technology to store IoT data. Authors in [2] proposed a blockchain-based framework to store IoT data off-chain. In the proposed framework, data generated by IoT devices is stored in DHTs and pointers to the stored data is written on the blockchain network. Moreover, in [21], authors proposed a blockchain-based architecture for smart home IoT applications, which uses cloud technology to store large-scale IoT data. [22] proposes Vegvisir, a partition tolerant blockchain for power-constrained IoT devices with limited network connectivity. A support blockchain is utilized to solve low storage capability of IoT devices. However, all these proposed frameworks considers a single storage solution and lacks auditable data provenance for large-scale IoT data. Moreover, all these proposed frameworks lack a middleware to dynamically assess and select the correct storage solution. Therefore, there is a need to propose a blockchain-based middleware to consider multiple storage technologies for large-scale IoT data storage along with trusted and auditable data provenance.

Recently, a lot of work has been done for a multi-cloud storage system [15], [16], [17], [18], [19], which decides the storage solution based on the requirement of application. SPANStore [15] places multiple replicas across multi-cloud, while minimizing the cost as a primary aim. The objective of a multi-cloud solution is to enhance security, availability, or to distribute the trust over multiple providers. Mcdb [16] proposes a multi-cloud database model to maximize the security of users data. Hybris [17] spreads the encrypted data over multi-clouds with an aim to maximize the confidentiality of the customer. However, these multi-cloud systems often target single objective optimization i.e. every system focus on specific application requirement at a time rather than considering multiple requirements. Moreover, these multi-cloud systems do not consider recent decentralized storage technologies which significantly differs from traditional cloud storage systems in term of performance and privacy. Furthermore, these multi-cloud systems focus on the dynamic behavior of cloud systems and do not consider the dynamic behavior of applications itself.

However, there exists a small number of multi-cloud papers which do consider applications with multiple requirements optimization goal [23], [14]. In particular, [23] proposes an adaptive middleware which only considers different cloud storage technologies. In this work, we are considering the the decentralized storage technologies along with cloud and multi-cloud systems. In addition the work only considers the dynamic behavior of cloud storage, while neglecting the dynamic behavior of applications. In our work, we have also considered the dynamic behavior of

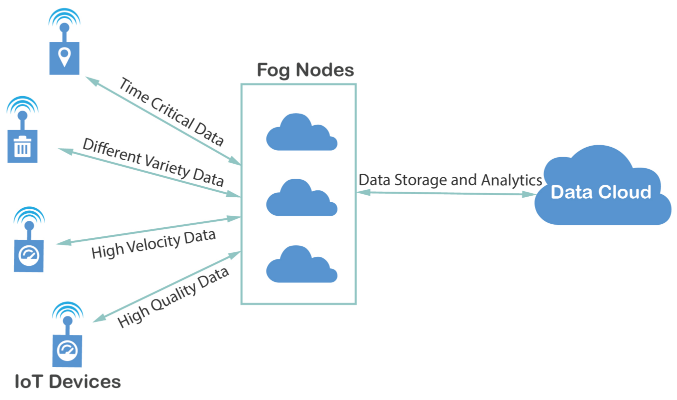


Figure 1: Traditional cloud architecture for IoT applications

IoT applications and their data access patterns. Moreover, the work lacks mathematical formulation of the problem. In this work, we formulate the data storage selection problem as a multi-objective optimization problem where the characteristics of the different storage solutions are captured using cost functions. Lastly, we have also integrated our adaptive middleware solution with blockchain network. The blockchain network can be used to store critical data on-chain and provides traceability and auditability and data integrity in our architecture.

III. IoT Architectures

In this section, we first review the popular fog and cloud based architecture for IoT applications. Then in section 3.2, we propose our first contribution, which is an extension to current IoT fog architecture to integrate decentralized storage technologies and blockchain for auditability, accountability and integrity.

A. Traditional Fog Architecture

Fig. 1 illustrates the overview of traditional cloud architecture for IoT devices. It consists of three layers i.e. IoT devices, fog nodes and the cloud. At the edge of the network, the IoT devices are used to monitor the environment and generates the raw data, which is sent to the fog node for pre-processing before being stored on the cloud premises. A fog/edge server, near to IoT devices, is a high-performance computing device which performs aggregation and filtering of raw IoT data. Fog nodes extract the time-critical IoT data and forward the archived data to cloud or multi-cloud premises for future storage and retrieval [1]. In multi-cloud the user's data is split across multiple cloud storage technologies to avoid vendor lock-in, better tolerate outages/failures along with better security [24]. Once the data has been pre-processed, the fog layer reports the output data to the cloud/multi-cloud and IoT device layer. As explained earlier, IoT applications have different service requirements, the current cloud architecture relies on a single cloud storage puts limits on the service requirements of IoT applications, which could be alleviated by another storage solution and therefore, this architecture is not suitable for IoT application with different service requirements.

B. Proposed Extension to Fog Architecture

In this paper we propose Blockchain-based Adaptive Middleware (BlockAM) architecture for IoT data storage. In this architecture, we are proposing an adaptive middleware to intelligently select the storage technology for IoT applications while considering multiple decentralized storage technologies along with the cloud. We also introduce blockchain to build trust in different network entities and to provide data integrity, auditability and accountability to IoT application users. The proposed architecture is shown in Fig 2.

Fog nodes: Unlike cloud computing, fog nodes operate at the network edge, near IoT devices, which assist in performing real-time computation and extraction of time-sensitive information from raw IoT data. A fog node could be any computational device between the IoT network and a back-end storage solution. Since fog nodes reside in the local network, we can assume that the communication between the IoT devices and the fog node is secure [25]. In our architecture, the fog node has three main responsibilities:

- Collects the data from the IoT devices and perform aggregation and filtering. Once the data processing is done, it forwards the data to the selected storage solution and keep track of IoT application-specific storage solution.
- Before sending the data to storage solution, it writes the hash of the data along with the IoT application ID to the blockchain network to ensure traceability and auditability.
- Keeps track of different service requirements of IoT applications and share the latest information to the middleware for data storage decision.

Blockchain network: The blockchain is a P2P network employed to store and process transactions in a trustless environment. It is a tamper-proof distributed ledger technology: once the information is written on the blockchain network, it can not be modified or removed. In our architecture, we use the blockchain to provide accountability and auditability for the IoT data and the storage selection decision. The blockchain network is directly connected to the fog nodes and the middleware. It is used to store the hash of the data and the storage decision. Moreover, the blockchain network is also used to store critical IoT data on-chain to provide 100% availability. The main aim of the blockchain network in our architecture is to build trust between the IoT network fog nodes and the middleware.

Adaptive middleware: The adaptive middleware is the core component of our architecture, which is directly connected to fog nodes, storage technologies and blockchain network. Our adaptive middleware solution,

- Continuously monitors the characteristics of storage technologies and maintains up-to-date service requirements for individual IoT application and use this information to solve the storage decision optimization problem.
- Records the storage decision and hash of the data to blockchain network to ensure data integrity, auditability and accountability properties of the middleware design.
- Defines the maintenance strategy for computational efficiency i.e. when to re-run the decision optimization problem to again decide the data storage solution for the IoT application.
- Manages the migration of data from one storage solution to another and predicts about future migrations based on past parameters.
- Provides the aggregation rate feedback to fog node intelligently to pre-process the data to optimize the data quality and precision.

However, in this work, we focus on the analytical problem formulation and blockchain aspect of the adaptive middleware design and heuristics algorithms to solve this problem.

Storage technologies: In the traditional architecture, the archived IoT data is stored in cloud/multi-cloud. The main objective of this work is to integrate different kind of storage technologies with cloud and multi-cloud architecture. Therefore, this part of our architecture represents the list of available storage technologies, which we can use to store large scale IoT data. In addition, we can also use the on-chain storage to store critical IoT data. As availability is the part of our cost model, if an IoT device is generating few bytes of data and if the service requirement of this data contains high availability, then we can directly store this data on-chain, subject to price requirements since on-chain storage is expensive. Moreover, since the fog node is aggregating the data, if some part of the aggregated data is critical, it can be added on-chain, while the other part of the aggregated data goes to the selected storage solution.

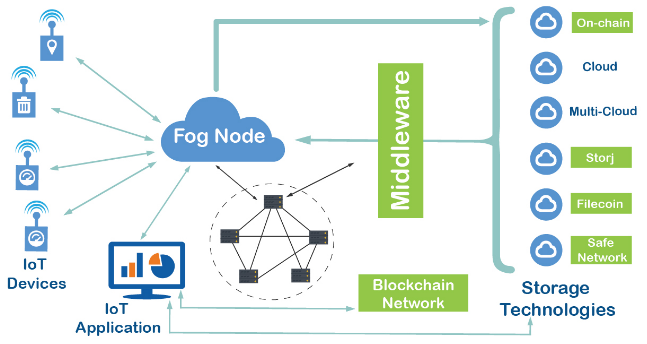


Figure 2: Middleware Architecture [26]

IV. Modelling the Cost of Decentralized Storage Technologies

In this section, we provide a comprehensive survey of decentralized storage networks and compare them in terms of security, performance and price etc. Moreover, we also estimate the analytical costs associated with each decentralized storage technology in terms of performance and price. To the best of our knowledge, no other survey has been reported on decentralized storage technologies.

Storj: Storj [9] is a cryptocurrency and a decentralized storage technology. The Storj network consists of clients, storage nodes and satellites. Storj is a partially decentralized network in which satellite nodes control the storage nodes and clients. Storj provides end-to-end encryption to the client's data and use erasure coding to add redundancy to ensure high availability. Storj creates chunks of client's data and distribute these chunks to the storage nodes all around the world. In order to compete with cloud solutions, to increase the scalability and performance of the network, Storj does not employ blockchain and byzantine distributed consensus and everything is controlled by Satellites. Since, the satellite maintains all the metadata regarding the storage nodes and clients, the performance cost in term of read and write latency can be represented as Equation 1:

$$Cost_p = (\frac{Size_{data}}{Bandwidth_{DL}}W_f + T) + (\frac{Size_{data}}{Bandwidth_{UL}}R_f + T) \quad (1)$$

where the first part of equation represents writing latency and the second part represents the reading latency. DL and UL represents bandwidth of downlink and uplink respectively. W_f represents the writing frequency, while R_f represents reading frequency of the data. T represents the response time of the storage servers.

Storj has a pricing scheme which is different for uploading and downloading data, i.e., clients have to pay to the Storj network to download their data. At the time of writing this paper, the Storj price is \$0.015 per GB per month to upload the data, while clients need to pay \$0.05 per GB to download their data. For example, if a client pay to the Storj network for 1 GB storage per month and the client retrieve the 1 GB data from the Storj network at the end of one month, client will pay \$0.015 + \$0.05 = \$0.065. Thus, the price cost can be represented as Equation 2:

$$Cost_{price} = \$0.015 Size_{data} + \$0.05 Size_{data} \quad (2)$$

where, the first and second part of the equation represents the writing and reading cost of the data respectively.

In the Storj network, the satellite nodes perform routine audits to maintain reputation for the storage nodes and verifies that the storage nodes are indeed storing the data. There are multiple satellites nodes in the Storj network: therefore, even if a satellite node is compromised, it only affects the storage nodes attached to it. Furthermore, Storj network does not provide anonymity to their clients. Storj network mitigates Sybil attack by employing Proof-of-Work (PoW) identity generation in S/Kademlia [27]. However, Storj is vulnerable to faithless satellites and storage node attacks where the storage nodes and satellites can show malicious behavior. In addition, Storj is not designed to protect against compromised clients.

Filecoin: Filecoin [10] is a decentralized storage network and works as an incentive layer over the InterPlanetary File System (IPFS) protocol [28]. Filecoin operates using a blockchain with Filecoin tokens, which are paid to the miners in return of providing storage to clients. Similarly, clients use Filecoin to hire storage nodes to store their data. Unlike Storj, the Filecoin network works without a trusted third party. Storage nodes publish their storage capacity as well as the price on the Filecoin blockchain network and the interested client can bid on this information. Once the client and the storage node agree on a deal, this deal is then saved in the Filecoin blockchain. Filecoin also provides end-to-end data encryption and can distribute the data to multiple storage nodes. Filecoin network has two markets: the storage market and the retrieval market. In the storage market, the Filecoin network adds two transactions to the blockchain network: the storage information publication by storage node and the signed deal. The retrieval market works using the gossip protocol to retrieve the client's data. Thus, the performance cost in term of read and write latency can be represented as Equation 3:

$$Cost_p = (\frac{Size_{data}}{Bandwidth_{DL}} + 2T_b + T)W_f + (\frac{Size_{data}}{Bandwidth_{UL}} + 1 + T)R_f \quad (3)$$

where the first part of equation represents writing latency and the second part represents the reading latency. $2T_b$ in the writing latency part represents the time taken by two transactions to be added in Filecoin blockchain. Similarly, the 1 in the reading latency part represents the additional latency by the gossip protocol in seconds. T represents the response time of the storage servers.

In the Filecoin network, every storage node decide the price for the storage by themselves. Moreover, the Filecoin pricing scheme is different for uploading and downloading data and it depends on the selected storage and retrieval node. As there is no defined storage and retrieval price and varies with each storage node, the price cost can be represented as Equation 4:

$$Cost_{price} = W_{price} Size_{data} + R_{price} Size_{data} R_f \quad (4)$$

where the first and second part of the equation represents the writing and reading cost of the data respectively. W_{price} represents the writing price of the data, while R_{price} represents the reading price. Moreover, the R_f factor in the reading price is the reading frequency factor. Unlike Storj, which offers reading cost per GB, the reading cost in Filecoin depends on the reading frequency with the specified data size.

In the Filecoin network, the storage nodes have to generate the Proof-of-Replication as well as Proof-of-Space-Time to make sure they have replicated the data and are storing the data for a predefined time. These proofs are posted on the Filecoin blockchain and other nodes on the network can check the validity of these proofs. These proofs mitigate the risk of Sybil attacks. However, the storage node can behave in a malicious way and with no proper replication of data, there is a risk of data unavailability. Filecoin does not provide anonymity to the users, however, they are planning to employ Zerocash [29] to resolve this issue.

Safe Network: The Safe network [11] aims to provide a decentralized peer-to-peer world wide web. Decentralized storage is therefore a major aspect of this network. Similar to Storj and Filecoin, Safe network also employs the free space of the storage nodes to store client data, using end-to-end encryption. The data is first converted to chunks and these chunks are then distributed to multiple storage nodes worldwide. Safe network has its own cryptocurrency called Safecoin, which is required to use applications on the decentralized Safe network. Safecoin does not use blockchain and has its own transaction verification mechanism to

Table I: Comparison of storage technologies

Storage	Decentralization	Smart Contract	Blockchain	Price Decision	Anonymity	Data Location
Cloud	Centralized	No	No	Cloud	No	Cloud Server
Storj	Partially Centralized	No	No	Storj	No	Multiple Storage nodes worldwide
Filecoin	Fully Decentralized	Yes	Yes	Storage Nodes	No	One or more storage nodes
Safe Network	Fully Decentralized	No	No	Safe Network	Yes	Multiple Storage nodes worldwide

reach consensus. Safe network has different client managers which are chosen randomly to manage client data. Upon reaching consensus that a client has enough safecoins to save the data, these client managers will distribute the client data to multiple storage nodes. Thus, the performance cost in term of read and write latency can be represented as Equation 5:

$$Cost_p = (\frac{Size_{data}}{Bandwidth_{DL}}+T)W_f+(\frac{Size_{data}}{Bandwidth_{UL}}+T)R_f \quad (5)$$

where the first part of equation represents writing latency while the second part represents the reading latency. The factor T represents the consensus time and response time between client managers in the network.

The Safe network decides the price of storage for the data. Moreover, the Safe network only charge users for uploading the data, while downloading the data is free of cost. At the time of writing this paper there is no defined storage price, therefore, the price cost can be represented as Equation 6:

$$Cost_{price} = W_{price}Size_{data} \quad (6)$$

where W_{Price} represents the cost of writing data to Safe network. The $Cost_{price}$ for Safe network just represents writing cost as there is no data reading cost for the clients.

The Safe network uses Proof-of-Resource to ensure that storage nodes are indeed storing the client data. It is done by an attempt to retrieve or store the data chunks randomly from/onto the corresponding storage nodes. The Safe network uses a zero-knowledge proof mechanism to perform this auditing. Moreover, the Safe network provides anonymity to its users. Since the client managers are chosen randomly, a collusion attack requires 88% of malicious clients to perform. Furthermore, the Safe network is vulnerable to malicious storage nodes.

Storage comparison:

Table I provides the comparison between cloud, Storj, Filecoin and Safe network in terms of decentralization, support for blockchain and smart contracts, storage price control, anonymity and the data location control. The specifications for cloud storage are derived from the literature [30]. In our proposed framework, all the public cloud storage technologies can be considered however, in this work, we are considering the Google cloud storage: the performance and the price cost for Google cloud can be represented as Equation 7 and Equation 8 respectively:

$$Cost_p = (\frac{Size_{data}}{Bandwidth_{DL}}W_f+T)+(\frac{Size_{data}}{Bandwidth_{UL}}R_f+T) \quad (7)$$

$$Cost_{price} = \$0.026Size_{data} \quad (8)$$

where, \$0.026 represents the cost of Google cloud storage per GB per month. Google cloud allows the retrieval of the data free of cost.

V. Adaptive Middleware Design

Our adaptive middleware solution continuously monitors the characteristics of storage technologies and select the storage solution based on the IoT application service requirements. The adaptive middleware is the core component of our architecture, which is directly connected to fog nodes as well as the storage technologies. We will describe the responsibilities of the adaptive middleware component and how it actually perform storage selection.

A. Problem Statement

System Model: We consider an IoT network consisting of ***D*** IoT applications, which continuously generates large amount of data with different data types I independently. The data generated by D IoT applications is first pre-processed at the fog node with an aggregation rate r. The aggregated data generated by the Fog nodes is then stored in different storage technologies. We consider ***J*** storage technologies to store the large scale IoT data, including public clouds (Amazon AWS, Google Cloud), decentralized storage technologies (Storj, Filecoin, Safecoin etc.) and blockchain network (on-chain storage) as shown in Fig. 2.

IoT applications D with different data types ***I*** have different service requirements [13]. Relying on a single cloud storage puts limits on the applications, which could be alleviated by another storage solution [14]. The middleware in the proposed architecture performs intelligent storage selection, as shown in Fig. 2. Adaptive middleware service, integrated with the fog and storage nodes, performs the selection of storage technology J given an IoT application D with multiple data types I subject to service requirements constraints.

Cost Minimization Problem Formulation:
IoT application $d \in D$ generates large amount of data, which is sent to fog node for pre-processing. Fog node produces $F_{d,i}$ bytes of data of data type $i \in I$, for upload to the storage technology. l_j represents the latency between the fog node and the storage technology j. $ping$ command can be used to dictate this delay and it depends on the geographic location of the storage server as well as fog node.

An adaptive middleware needs to decide which storage technology should be selected in order to jointly minimized the incurred monetary cost as well as Quality of Service (QoS) parameters cost. The total cost C_t, which needs to be jointly minimized has five ingredients: latency cost, storage and retrieval cost, bandwidth cost, availability cost and privacy cost.

Decision Variable: A binary decision variable y_j is formulated which corresponds to the selection of storage technology for IoT application $d \in D$ with data type $i \in I$. The decision variable y_j, $\forall\ j \in J$, indicates whether the storage solution j is selected for selected IoT application (y_j = 1) or not (y_j = 0). We require that for every decision model solution, j storage solutions should be selected for data types i of IoT application. Moreover, same storage solution j can be selected for more than one data type i. Let $\overline{y}$ = $(y_j)_{\forall j \in J}$ be the set of all possible storage variables and it is given by:

$\mathcal{Y}$={ $\overline{y} \mid \sum_{j \in J} y_j \leq J$ and $y_j \in \{0,1\}$, $\forall j \in J$ }

Overall Cost: The overall cost which needs to be jointly minimized contains the following elements:

(1) The ***bandwidth cost*** refers to the price set by storage technologies for uploading and downloading the data. If the total amount of data generated by an IoT application, that needs to be stored on storage technology is $\sum_{i \in I}(F_{d,i})$ and P_j is the bandwidth price set by the storage technology j, the overall bandwidth cost can be represented as Equation (9),

$$C_{BW}(\overline{y}) \triangleq \sum_{j \in J}\sum_{i \in I} y_j P_j F_{d,i} x_i, \;\; \forall i \in I, j \in J, \forall j \in J \exists! i : x_i \quad (9)$$

where $x_i \in \{0,1\}$ represents whether the selected data type i is considered for the storage technology j, while the condition $\forall j \in$

$J\exists!i : x_i$ states that only one storage solution must be selected for one data type I of an IoT application.

(2) The ***storage and retrieval cost*** is the monetary cost set by the storage technologies for storing and retrieving the data for specified time period. Let the total amount of data that needs to be stored on storage technology is $\sum_{i\in I}(F_{d,i})$, while the amount of data that needs to be retrieved is $\sum_{i\in I}(R_{d,i})$. Let Gj and Hj represents the per byte storage and retrieval cost for j^{th} storage technology respectively. The values of G_j and H_j is set by the respective storage technology. The overall storage cost can be represented as Equation (10),

$$C_{CS}(\overline{y}) \triangleq \sum_{j\in J}\sum_{i\in I} y_j(GjF_{d,i} + HjR_{d,i})x_i,\ \ \forall i\in I, j\in J,\ \ \forall j\in J\exists!i : x_i \tag{10}$$

(3) The ***latency cost*** represents the latency of the storage technology i.e. how much time is required to upload the data to the storage technology. Moreover, the QoS performance metrics i.e. l_j and T_b are also considered for delay sensitive IoT applications. l_j and T_b corresponds to the link latency and block time respectively. The overall latency cost of incurred latency to upload the overall data $\sum_{i\in I}(F_{d,i})$ can be represented as Equation (11),

$$C_{LT}(\overline{y}) \triangleq \sum_{j\in J}\sum_{i\in I} W_L(\frac{F_{d,i}}{BW_j}y_jx_i + l_j + T_b),\ \ \forall i\in I, j\in J,\ \ \forall j\in J\exists!i : x_i \tag{11}$$

where W_L represents the weight to convert the latency cost to monetary cost, while BW is the bandwidth provided by j_{th} storage technology. The overall latency cost depicts that high latency between fog node and the storage server brings high cost. It should be noted that the value of T_b is only assigned to blockchain-based storage technologies and is 0 otherwise.

(4) The ***availability cost*** represents the availability of the storage technology i.e. the probability that the storage server will be online whenever the IoT application owner requests the data. Availability of any system is represented in terms of probability [31]. For IoT applications with high availability requirement, this cost needs to be maximized. The overall cost of availability can be represented as Equation (12),

$$C_{AV}(\overline{y}) \triangleq \sum_{j\in J} W_A y_j \alpha_j,\ \ \forall j\in J \tag{12}$$

where W_A represents the weight to convert the availability cost to monetary cost, while α_j represents the availability of j^{th} storage technology in terms of probability i.e. $0 \le \alpha_j \le 1$. The availability cost is assumed to be determined by the availability SLA defined by the storage technologies.

(5) The ***privacy cost*** represents the whether the required storage technology offers privacy or not. Few IoT applications e.g. IoT healthcare requires privacy therefore, this cost needs to be maximized. Let $\beta_j \in \{0,1\}$ denotes the privacy parameter for a given storage technology j i.e. β_j = 1 if the storage technology offers privacy otherwise 0. The overall privacy cost can be represented as Equation (13),

$$C_{PR}(\overline{y}) \triangleq \sum_{j\in J} W_P y_j \beta_j,\ \ \forall j\in J \tag{13}$$

where W_P represents the weight to convert the privacy cost to monetary cost.

Optimization Problem: The objective function that needs to be minimized consists of the bandwidth cost, storage and retrieval cost and latency cost i.e. $minimize(C_{BW}+C_{CS}+C_{LT})$, while the availability and privacy cost needs to be maximized i.e. $maximize(C_{AV}+C_{PR})$. To jointly minimize the overall cost, the total cost can be represented as Equation (14),

$$\mathbb{C}(\overline{y}) = C_{BW}(\overline{y}) + C_{CS}(\overline{y}) + C_{LT}(\overline{y}) - C_{AV}(\overline{y}) - C_{PR}(\overline{y}) \tag{14}$$

Thus, we formulate our optimization problem by uniting these multi-objective functions into a single objective function as Equation (15a),

$$\begin{aligned}
\min \quad & \mathbb{C}(\overline{y}) && \text{(15a)}\\
\text{subject to} \quad & \overline{y}\in\mathcal{Y}, && \text{(15b)}\\
& \sum_{j\in J}\sum_{i\in I} P_jF_{d,i} \le \mathcal{B}, \forall i\in I, \forall j\in J, && \text{(15c)}\\
& \sum_{j\in J}\sum_{i\in I}(GjF_{d,i} + HjR_{d,i}) \le \mathcal{M}, \forall i\in I, j\in J && \text{(15d)}\\
& \sum_{j\in J}\sum_{i\in I}(\frac{F_{d,i}}{BW_j}y_jx_i + l_j + T_b) \le \mathcal{L}, \forall j\in J && \text{(15e)}\\
& \alpha_j \le \mathcal{A}, \forall j\in J && \text{(15f)}\\
& \beta_j \le \mathcal{B}, \forall j\in J && \text{(15g)}
\end{aligned}$$

where, $\mathcal{L}$, $\mathcal{A}$ and $\mathcal{B}$ are the maximum allowed cost for latency, availability and privacy respectively. $\mathbb{C}(\overline{y})$ is given in Eqn. (14). Constraint (15c) states that the total bandwidth cost should be less than the defined bandwidth budget $\mathcal{B}$. Similarly, (15d) states that the total monetary cost should not exceed the maximum allowed monetary budget $\mathcal{M}$ for storage and computation. Constraints (15e), (15f) and (15g) ensures that the cost of latency, availability and privacy should be within defined thresholds i.e. $\mathcal{L}$, $\mathcal{A}$ and $\mathcal{B}$ respectively. It should be noted that these constraints are the service requirement of IoT applications defined in the middleware and are unique for each IoT application.

The data centers selection and placement problems have been shown to be related to the Facility Location Problem (FLP) [32], [33]. In a typical FLP, a set of facilities and clients demands are given, and the goal is to select and open the facilities to serve clients based on different demands, in order to minimize the overall cost of facility opening and service [32]. Uncapacitated facility location (UFL) [34] problem is a metric variant of FLP in which there is no capacity limitation on the facilities and a well known NP-hard problem [35]. Our optimization problem is similar to the metric UFL problem in which the storage technologies represents the facilities and the service requirements of IoT applications corresponds to the demands requested by the clients. As the UFL problem is NP-hard, our optimization problem is also NP-hard.

Theorem 1. The cost minimization optimization problem (15a) is NP-hard.

Proof: Motivated by [36], we prove our optimization problem to be NP-hard. We construct a polynomial-time reduction to our cost minimization optimization problem (15a) from the uncapacitated facility location problem (UFL), a well known NP-hard problem [35]:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} f_iz_i + \sum_{i=1}^{m}\sum_{i=1}^{n} c_{i,j}u_{i,j} && \text{(16a)}\\
\text{subject to} \quad & \sum_{i=1}^{n} u_{i,j} = 1, \forall i\in I && \text{(16b)}\\
& u_{i,j} \le z_i, \forall i\in I, \forall j\in J && \text{(16c)}\\
& z_i, u_{i,j} \in \{0,1\}, \forall i\in I, \forall j\in J && \text{(16d)}
\end{aligned}$$

Similar to the UFL problem, our optimization problem consist of fixed and variable cost elements. The fixed cost component are comprised of C_{AV}, C_{PR} and can be represented as C_{fixed} = $(\sum_{j\in J} W_A\alpha_j + W_P\beta_j)y_j$, while C_{BW}, C_{CS}, C_{LT} represents the variable cost and can be written as $C_{var} = \sum_{j\in J}\sum_{i\in I}(P_jF_{d,i}x_i + (Gj + Hj)F_{d,i}x_i + W_Ll_jF_{d,i}x_i)y_j$. However, in our optimization problem, we are maximizing the fixed cost as compared to the minimization in the UFL problem. Thus, we modify our optimization problem in Equation. 15a by introducing the two new parameters i.e. $\hat{C_{AV}}, \hat{C_{PR}}$. Compared to Equation (12), $\hat{C_{AV}}$ corresponds to the cost on $non-availability$ and can be represented as Equation (17),

$$\hat{C_{AV}}(\overline{y}) \triangleq \sum_{j\in J} W_A y_j \hat{\alpha_j}, \ \forall j \in J \tag{17}$$

where $\hat{\alpha_j}$ represents non-availability value i.e. $\hat{\alpha_j}$ = 1 - α_j. Similarly, $\hat{C_{PR}}$ represents the cost of $no-privacy$ as compared to Equation (13) and can be represented as Equation (18)

$$\hat{C_{PR}}(\overline{y}) \triangleq \sum_{j\in J} W_P y_j \hat{\beta_j}, \ \forall j \in J \tag{18}$$

where $\hat{\beta_j}$ represents no-privacy value i.e. $\hat{\beta_j}$ = 1 - β_j. In contrast to the fixed costs C_{AV}, C_{PR}, we want to minimize the $\hat{C_{AV}}, \hat{C_{PR}}$ costs. These costs are actually the two sides of the same coin i.e. maximizing the availability and privacy is similar to minimizing the non-availability and no-privacy respectively. Therefore, we can write our optimization problem as Equation 19a,

$$\min \quad \sum_{j\in J}(\hat{C_{AV}} + \hat{C_{PR}}) + \sum_{j\in J}\sum_{i\in I}(C_{BW} + C_{CS} + C_{LT}) \tag{19a}$$

$$\text{subject to} \quad \overline{y} \in \mathcal{Y}, \tag{19b}$$

$$\sum_{j\in J}\sum_{i\in I} P_jF_{d,i} \leq \mathcal{B}, \forall i \in I, \forall j \in J, \tag{19c}$$

$$\sum_{j\in J}\sum_{i\in I}(GjF_{d,i} + HjR_{d,i}) \leq \mathcal{M}, \forall i \in I, j \in J \tag{19d}$$

$$\sum_{j\in J}\sum_{i\in I}(\frac{F_{d,i}}{BW_j}y_jx_i + l_j + T_b) \leq \mathcal{L}, \forall j \in J \tag{19e}$$

$$\hat{\alpha_j} \leq \hat{\mathcal{A}}, \forall j \in J \tag{19f}$$

$$\hat{\beta_j} \leq \hat{\mathcal{B}}, \forall j \in J \tag{19g}$$

Given an instance I of the UFL problem, where $I = (m, n, c_{ij}, f_i)$, we map it to a problem instance of our cost minimization optimization problem (19a) with $\hat{I} = (|J| = n, |I| = m, C_{var} = c_{ij}, C_{fixed} = f_i, y_i = u_{i,j})$. It can be seen that the mapping of problem instance I to $\hat{I}$ can easily be done in polynomial time [36]. Thus, if some algorithm solves the cost minimization optimization problem (15a) with problem instance $\hat{I}$, it also solves the related UFL problem with problem instance I as well. Thus, we can consider the UFL problem as a special case of our cost minimization optimization problem. As the UFL problem is NP-hard, our cost minimization optimization problem must be NP-hard as well.

B. Blockchain for auditability and accountability

As discussed previously, auditability and accountability is important to ensure trust and confidence, while operating in an untrusted environment. The blockchain component in the proposed architecture enables auditability and accountability of the storage decision. Every time, the middleware is making a decision about the storage solution, it records the IoT application ID along with the data ID and the storage decision on the blockchain. The application ID represents the type of IoT application for which the data is being stored, while the data ID represents the specific data range of that particular IoT application ID. This stored information allows the IoT application owners to audit the middleware decision while retrieving the data from storage technologies. Moreover, this decision storage also provides accountability and auditability of storage technologies if they delete or tamper with the user data. Besides storage decision, the middleware also stores the migration decision from one storage solution to the other. The migration decision includes the previous storage solution, the current storage solution and ID of the data being migrated. This ensures the auditability and accountability of the decisions made by the middleware design.
Usability of the Stored IoT Data: In order to retrieve the archived data with specific data ID, the IoT application user will ask middleware for the corresponding storage decision. Once the decision is known, the IoT application user will retrieve the data from the storage solution. The storage solutions provides user friendly interface to the IoT application user to store and retrieve data therefore, once the data has been retrieved, the hash of the data will be matched with already stored hash in the blockchain network. If the hash of the data matches with the current data hash, the data will be considered valid.

VI. Proposed Heuristic Algorithms

In this section, we propose two polynomial-time heuristic algorithm to approximate a solution to the NP-hard optimization problem formulated in Section V-A.

We first present a dynamic programming based heuristic algorithm to approximate a solution to the proposed optimization problem. We then present a greedy-style based heuristic algorithm to feasibly select the storage technology for IoT application. The output of these heuristics is to select a storage technology for large scale data generated by the IoT applications. In rest of this section, we provide the details of these heuristic algorithms.

A. Dynamic Programming (DP) based Heuristic

We first propose the dynamic programming based heuristic to solve the optimization problem formulated in Equation. (15a). This algorithm solves (15a) by computing the corresponding parameters cost (Equation. (9) - (13) for different storage technologies and selects $\overline{y}$ based on the first fit strategy. Using Dynamic Programming, we construct a 2-D table $T(y_j, C_j(\overline{y}))$, which contains the minimum cost values for different parameters of storage technologies. The table $T(y_j, C_j(\overline{y}))$ has y_j rows, which corresponds to the j storage technologies available to be selected by IoT application, and $C_j(\overline{y})$ columns corresponding to the cost of dynamic parameters of storage technologies derived in Equation. (9) - (13). Algorithm 1 summarizes the design of heuristic to solve the optimization problem (15a). The algorithm starts with $\overline{y} = 0$ along with $T(y_j, C_j(\overline{y}))$ = 0 and iteratively solves and selects the storage technology for m IoT applications data types. At start of loop at line 5, the dynamic parameters cost is calculated for j^{th} storage technology. Given the service requirement for a specific IoT application data type as $Eqn\ (15c) - (15g)$, line 7 checks if the calculated cost parameters matches with the service

Algorithm 1 Dynamic Programming based Heuristic
```
1:  Initialize $\overline{y}, T(y_j, C_j(\overline{y}))$
2:  $\overline{y} \leftarrow 0$
3:  $T(y_j, C_j(\overline{y})) \leftarrow 0$
4:  for $x \leftarrow 1$ to $i$ do
5:      for $y \leftarrow 1$ to $j$ do
6:          Find all the parameters costs associated with $y_j$
7:          if $parameters\ cost$ ¡ $Eqn\ (15c) - (15g)$ then
8:              Assign $y_j$ storage technology to $x_i$ IoT application
9:              Store the parameters cost value in $T(y_j, C_j(\overline{y}))$
10:             $break$
11:     Find $\min_{\forall \overline{y})\in\mathcal{Y}}$T(y$_j, C_j(\overline{y}))$ s.t. $Eqn\ (15c) - (15g)$
12:     if $no\ storage\ solution\ found$ then
13:         Go to step 5
14: end
```

requirements. If the service requirement for certain IoT application matches with the j^{th} storage technology, it assigns the same storage solution to current IoT application data type at line 8. After the assignment, these cost parameters are stored in the table against the current storage technology. The algorithm then goes to line 11 to select storage solution for next IoT application data type by matching the current service requirements with available storage technology parameters in the table. If no suitable storage technology is available for current IoT application, the algorithm then goes back to step 5 on line 13 to find a new storage solution.

The main rationale behind the dynamic programming approach is that, we are dividing our main optimization problem into independent sub-problems in which the storage technology for each IoT application is chosen separately. Moreover, while calculating the solution for the sub-problems, we utilize the already calculated parameters for the future sub-problems [37]. Therefore, when a new storage technology is selected for specific IoT application, the cost of dynamic parameters associated with the storage technology is stored in the table using memoized top-down approach. Thus, when the selection process is repeated for a new IoT application, the algorithm first checks the table for possible storage technology that fulfills the service requirements. If found, the algorithm does not need to calculate the cost parameters of storage technology again. Similarly, more IoT applications storage decision will result in more entries in the table thus, leads to efficient storage selection.

Theorem 2. Algorithm 1 provides a feasible solution to the optimization problem (15a) in polynomial-time.

Proof: For the optimization problem (15a), the constraints in Eqn. $(15c) - (15g)$ are satisfied by line 7 in the algorithm. The feasible storage solution y_j will not be selected in line 8, until it satisfies all the service requirements for x_i IoT application in line 7. Furthermore, once the table is updated in line 9, the storage solution will be searched for x_{i+1} IoT application in line 11. If no feasible storage solution is found in the table, the algorithm will again go to line 6 to find the feasible solution for x_{i+1} IoT application and update the table accordingly. Thus, the Algorithm 1 provides a feasible solution to all the IoT applications.

For the time complexity of Algorithm. 1, the loop in line 4 runs at most i times for all the IoT applications. Similarly, the loop in line 5 runs j times in a worst case scenario for all the storage technologies. In addition, the line 6 runs for the k number of service requirements. Thus, we conclude that the worst case time complexity of Algorithm. 1 is $O(ijk)$, which is polynomial-time.

B. Greedy-Style (GS) Heuristic

In this section, we present a greedy-style heuristic algorithm to solve the optimization problem (15a). In some IoT applications scenario, only few important service requirement might be defined by the application owner. For instance, the IoT health application owner might define privacy as the only service requirement. Therefore, in this case it is not feasible to calculate the unnecessary cost parameters related to storage technologies. Thus, we propose a heuristic algorithm, which only takes into consideration the storage technology cost parameters associated with the IoT application service requirements. The proposed algorithm selects the storage technology subject to the service requirements of IoT application using first-fit strategy. The rationale behind the algorithm is that it randomly takes the storage decision as soon as certain number of defined optimization conditions (Eqn. $(15c) - (15g)$) are fulfilled by the dynamic cost parameters of storage technologies. Algorithm. 2 illustrates the greedy-style heuristic for the selection of storage technology.

We define the term γ in our algorithm, which defines the service requirements as the conditions (Eqn. $(15c) - (15g)$) that should be fulfilled before selecting a storage technology. For example, if latency and privacy are the only service requirement of a certain IoT application, the value of γ will be a tuple of 2 parameters i.e. <latency,privacy>. The algorithm starts by initializing the values of number of IoT applications and their respective γ parameters. For each IoT application data type m, line 6 calculates the associated cost parameters of storage technologies. These cost parameters are then matched with the γ service requirements of IoT application data type m in line 7. If the γ conditions meet in line 8, the storage solution is selected for IoT application m in line 9.

Algorithm 2 Greedy-Style Heuristic

```
1: Initialize $M$
2: Initialize $m \leftarrow 1$
3: Initialize $\gamma \leftarrow \gamma_1, \gamma_2, ..., \gamma_m$
4: while $m \neq M$ do
5:     for $y \leftarrow 1$ to $n$ do
6:         Find $\gamma_m$ parameters costs associated with $y_n$
7:         Match parameters costs with $\gamma_m$ Eqn. $(15c) - (15g)$
8:         if $\gamma_m$ conditions are fulfilled then
9:             Assign $y_n$ storage technology to $m_{th}$ IoT application
10:            $break$
11:    m = m + 1
```

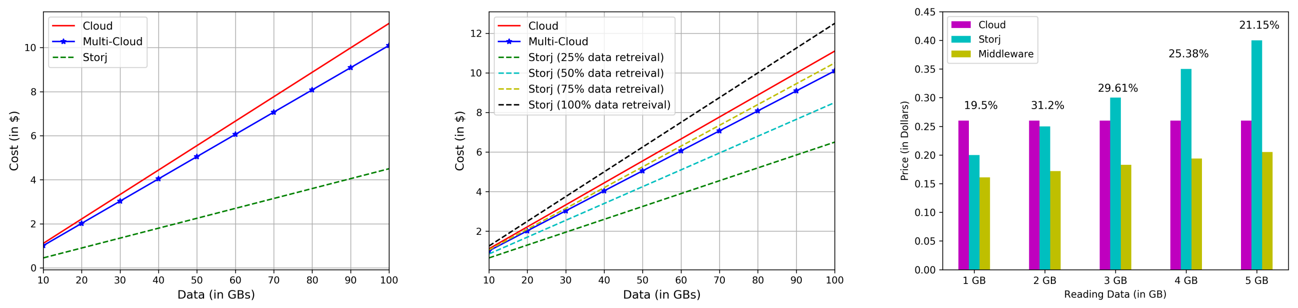
Theorem 3. Algorithm 2 provides a feasible solution to the optimization problem (15a) in polynomial-time.

Proof: Fewer number of γ service requirements generates the solution fast but the solution may be far from optimal because the optimization problem (15a) is solved by fulfilling the γ number of conditions (γ). However, since the IoT application owner is only interested in γ service requirements, the achieved solution will be feasible by fulfilling γ service requirements, while ignoring the unnecessary service requirements optimization problem (15a). Therefore, Algorithm 2 provides a feasible solution to the optimization problem (15a), while solving for γ conditions. For the time complexity of Algorithm. 2, the while loop runs for m times, while the for loop runs for n times in the worst-case scenario. Thus, the worst-case overall computational complexity of Algorithm 2 is $O(mn)$, which is polynomial-time.

Usability of Algorithms: It is not always necessary to consider all the service requirement i.e constraints in the multi-objective optimization problem to decide the storage solution for a particular IoT application. It rather depends on the requirements specified by the IoT application owners. For example, if the IoT application owner only specify the minimum budget requirement and does not specify any other requirement, the multi-objective optimization problem defined in Eqn. (15a) will only consider the constraints (15c) and (15d). In this case, we argue in the favor of Greedy-Style heuristic because it allows the IoT application owner to specify the service requirements for individual applications. However, if an IoT application considers all the service requirements in the optimization problem defined in Eqn. (15a), we recommend to use the DP-based heuristic because of its performance and accuracy as described in Section. VIII

VII. Implementation

This section presents the implementation details of our middleware architecture. In order to evaluate our middleware design, we have implemented our own prototype design in Python programming language, which allows us to perform baseline comparison with our proposed algorithms as well as to analyze the impact of different IoT parameters on our overall optimization cost. To mainly focus on the evaluation of middleware part in our overall design, we implemented five main functionalities in middleware script i.e. solving the optimization problem to decide storage solution, writing the storage decision to the blockchain network, maintaining the decision in local database, continuous monitoring of performance metrics of storage solutions and storing the service requirements of individual IoT applications. In order to generate the IoT application requirements, we implemented the fog node functionality in our middleware design in Python programming language.



(a) Overall cost with no data retrieval (b) Overall cost with data retrieval (c) Price Comparison

Figure 3: Dynamic properties of maintenance strategy and migration plan

We implement blockchain functionality through Go language implementation (Geth[1]) of Ethereum blockchain platform. Moreover, we have written a smart contract using the Solidity programming language. Fog node and the middleware is communicating with the Ethereum node using web3py[2], a Python library to interact with Ethereum network. In the implementation of storage technologies to test our middleware, we have chosen Google cloud storage and Storj. The main reason to choose these two storage solution for our implementation is because they provide concrete pricing for their network. In order to communicate between the Python fog node and google drive, we have used Google Drive API[3]. Moreover, we have used Storj test network[4] in our prototype implementation.

In this implementation, the interaction between the fog node, middleware, blockchain and the storage technologies. The workflow of the system can be described as following steps.

- The fog node generates the IoT application parameters and send it to the middleware to decide which storage solution to choose. At the same time, fog node stores the hash of the data along with Application ID on the blockchain network.
- Middleware takes these IoT application input parameters, and take the monitored storage characteristics and solves the optimization problem to find the storage technology for the IoT application based on the stored service requirements. Moreover, it stores the decision in the local database as well as in the blockchain network.
- After receiving storage decision from middleware, the fog node uploads the data to the selected storage node.

VIII. Evaluation And Analysis

In this section, we evaluate our adaptive middleware solution. We first compare the exact overall optimization cost for different storage technologies to show the impact of IoT application parameters on the optimization decision. We also compare the prices of multiple storage technologies for a set of IoT applications parameters with the price solution generated by our middleware design. The goal of this analysis is to assess how much cost we can save by employing our middleware design. Moreover, we also compare the solution of our two heuristic algorithms explained in Section VI with the baseline exact solution. In addition, we also perform the performance and scalability analysis of our algorithms for large number of IoT applications as well as storage technologies. The goal of this analysis is to assess the accuracy and viability of our proposed algorithms in middleware design. In addition, we provide the gas cost estimation for the smart contracts functions, written to interact with middleware and fog node.

A. System Setup

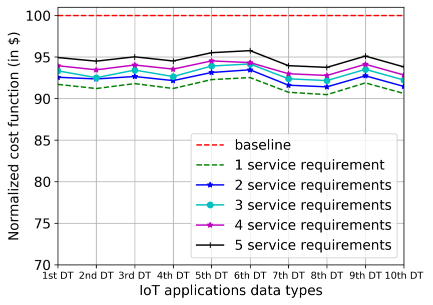
Our experiments are run on UBUNTU 16.04 desktop with 8^{th} Generation Intel $Core^{TM}$ i7-8650U Processor with 16 GB RAM. The Python based fog node and the middleware are running on the same machine. The interaction between the middleware and fog node is done by importing middleware functionality in main fog node client.

Simulation Parameters: Since the development of decentralized storage technologies is in its infancy and the parameters associated to these storage technologies are not publicly available, we only consider the real-world parameters of cloud and storj in our experiments because of their public availability. We consider Google cloud as our cloud storage technology and the charges for storing one GB of data to Google cloud is set to 0.026 $/GB [5]. The bandwidth cost of Google cloud is set to 0.085 $/GB. Based on the Google cloud SLA, the availability value is set to 99.99%[6] and the privacy value is set to 0 as Google cloud offer no privacy. Similarly, the cost to store one GB of data to Storj is set to 0.015 $/GB[7], while the bandwidth cost is set to 0.03 $/GB[8]. Storj also charges customer for retrieving the data therefore we set the data retrieval cost to 0.05$/GB. The availability cost is set to 99.99%[9] and the privacy value is set to 0. The link bandwidth is set to 100 Mbps, while the block time is set to 0 for both storage technologies. The default value of transforming the delay, availability and privacy to monetary cost is set to 0.01.

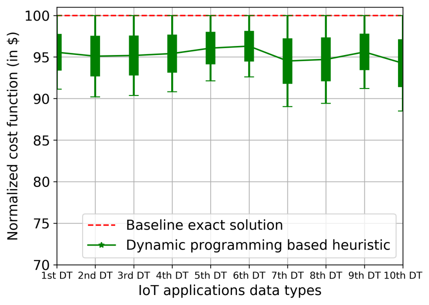
B. Results

1) Impact Of IoT Application Parameters: We first investigate the impact of IoT application parameters on the overall cost of the optimization problem formulated in Equation. (15a). We compare the exact overall cost for different IoT data size values, that need to be stored and retrieved, for cloud, storj and multi-cloud storage technologies. Fig. 3a illustrates the overall cost of multiple storage technologies with respect to different data sizes with no data retrieval requirement. It can be seen that the cost of storj is much lower as compared to cloud and multi-cloud because storj offers less storage and bandwidth cost as compared to cloud and multi-cloud. Similarly, Fig. 3b shows the overall cost with some percentage of data retrieval requirement. It can be seen that the overall cost of storj is increasing with the increasing percentage of data retrieval requirement. This is due to the fact that storj charges for data retrieval as well as bandwidth cost associated with it. It can be

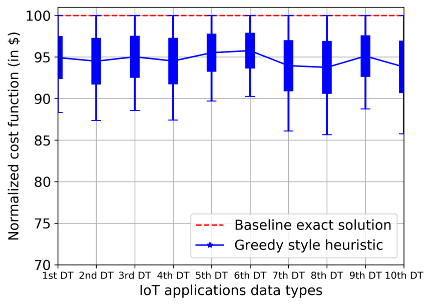
[1]https://github.com/ethereum/go-ethereum/wiki/geth
[2]https://web3py.readthedocs.io/en/stable/
[3]https://developers.google.com/drive/
[4]https://storj.io/blog/2019/01/getting-started-with-the-storj-v3-test-network-storj-sdk/
[5]https://cloud.google.com/storage/pricing
[6]https://cloud.google.com/compute/sla
[7]https://storj.io/
[8]https://storj.io/storage-node-estimator/
[9]https://storj.io/blog/2019/08/the-role-of-qualification-gates-in-getting-to-beta-and-beyond/



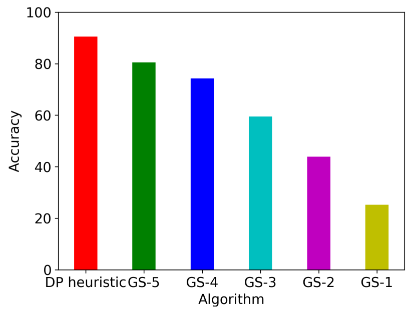
(a) GS heuristic cost comparison



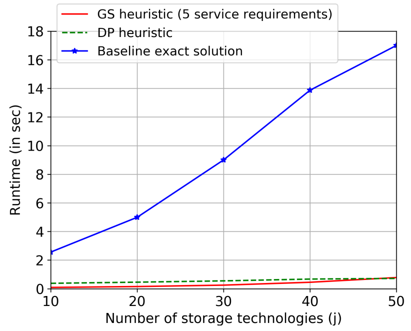
(b) DP heuristic comparison



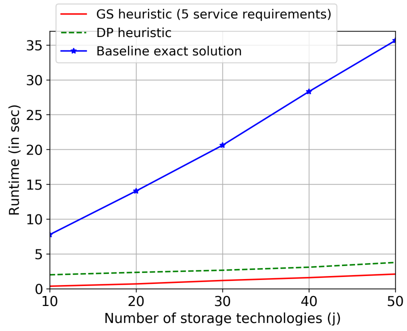
(c) GS heuristic comparison



(d) Accuracy comparison



(e) Best runtime comparison



(f) Worst runtime comparison

Figure 4: Heuristics comparison

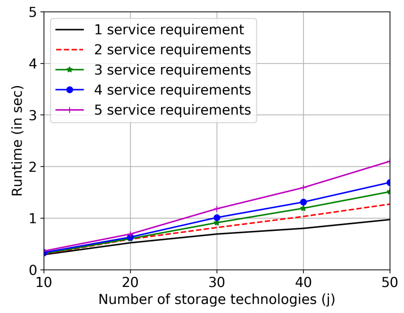
seen from that the overall cost of storj surpass the cost of cloud for high data retrieval rate. This implies that the data storage and retrieval size makes an impact on the storage technology decision for IoT application.

2) Price Comparison: Fig. 3c shows the experiment results for the Cloud and Storj only solution along with the adaptive middleware solution generated from greedy-style heuristic by keeping price as the only service requirement. The data has been divided into 5 data types with 1GB data size each, along with individual data retrieval requirement. The experiments are performed with writing data size of 5GB, while the reading data size ranges from 1GB - 5GB. It can be seen that for up to 2 GB retrieval data, cloud is the most expensive solution among all, while our adaptive middleware solution is providing least cost. However, while performing experiment with up to 5 GB reading data size, it can be seen that storj cost surpass the cloud cost thus, making it the most expensive solution. However, in all the cases, our adaptive middleware solution offers least price. Fig. 3c also shows the cost reduction percentage for our middleware design. The cost reduction percentage is calculated by the ratio of total cost offered by the middleware solution with the total minimum cost offered by cloud or storj. It can be seen that the Middleware reduces cost up to 31.2% for the reading 2GB data. The percentage then starts to decrease because reading more data incurs more cost in case of Storj thus, making cloud cheaper solution. It can also be seen that when the data retrieval requirement is almost 50% of the data storage requirement, storj becomes an expensive solution than cloud.

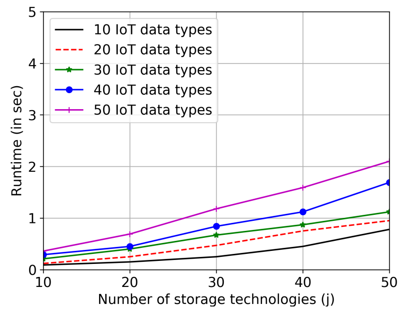
3) Heuristics Comparison: We compare our two proposed heuristic algorithms against a baseline exact solution for middleware data placement decision. We generate the IoT application data type requirements randomly and solves the decision problem through real-world storage parameters of cloud, storj and multi-cloud. Since, the generation of IoT application requirements and selection of storage technologies in Algorithm 1 and 2 are performed randomly, we repeat each simulation at least 1000 times to generate acceptable average comparison results for the randomly generated IoT application requirements. We compare the cost of greedy style heuristic algorithm with the baseline solution for different service requirements in Fig. 4a for 10 individual IoT applications with randomly generated service requirements. It can be seen that the overall cost is minimized if the algorithm considers all the service requirements. However, the cost increases with the decrease in the considered service requirement of greedy heuristic algorithm. This is due to the fact that for few service requirement, the algorithm will take decision as long as the defined service requirements are met thus, results in more cost.

Fig. Fig. 4b and Fig. 4c show the comparison of two heuristic algorithms with the baseline exact solution along with the min-max and standard deviation of the cost function of dynamic programming based heuristic and greedy-style heuristic respectively. We observe that the cost incurred by dynamic programming based heuristic is close-to-optimum exact solution and outperforms the greedy-based heuristic in terms of cost reduction. This is because when the cost table is completed, the dynamic programming based heuristic will always select the storage technology with minimum cost compared to random storage selection in greedy-style heuristic, which results in cost reduction. We further investigate the accuracy of two algorithms in term of exact decisions in Fig. 4d. The accuracy of the algorithm is calculated by comparing the algorithm storage decisions with the optimal storage decisions, while repeating the experiments several time and averaging the results. It can be seen that dynamic programming based heuristic achieves approximately 92% accuracy as compared to maximum accuracy of around 80% by greedy-style based heuristic. It can be observed that the accuracy of greedy style based heuristic with few service requirement is low. However, the accuracy is calculated by comparing the greedy style algorithm decisions with the exact solution ones. Even with low accuracy, the algorithm always fulfills the application service requirement defined by the IoT application owner, while ignoring the undefined ones.

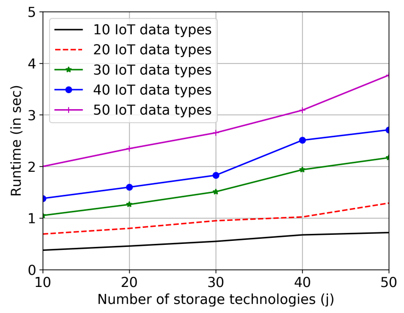
4) Computation Time: We then investigate the best and worst case runtime comparison of the proposed algorithms with the baseline exact solution in Fig. 4e and Fig. 4f respectively. The proposed algorithms



(a) GS heuristic service requirements runtime

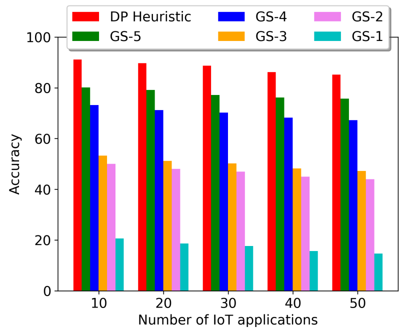


(b) GS heuristic IoT data type runtime

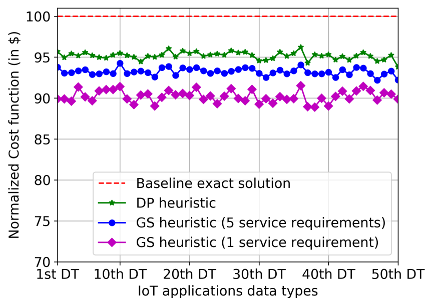


(c) DP heuristic runtime

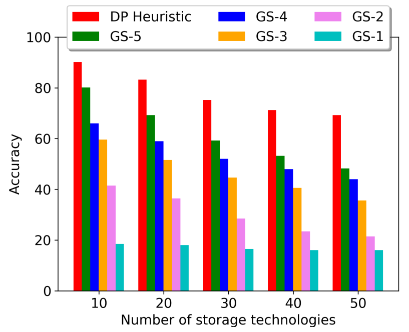
Figure 5: Runtime comparison of proposed heuristics



(a) Accuracy comparison



(b) Cost comparison



(c) Accuracy comparison

Figure 6: Sensitivity analysis of storage technologies

Table II: Gas cost estimation for smart contract functions

Function	Gas Units	Gas Price (in Wei)	Gas Cost (in Wei)	Gas Cost (in Ethers)	Avg. Confirmation Time (in sec)
storeHash	40290	1000000000	40290000000000	0.00004029	42.6
storeDecision	60311	1000000000	60311000000000	0.00006031	42.6
migration	60917	1000000000	60917000000000	0.00006091	42.6

are run for 10 IoT application data types with all service requirement for best case scenario, while for the worst case scenario, both algorithm are run for 50 IoT applications. Moreover, the runtime is calculated by repeating the experiment several time and taking the average value. It can be seen that the greedy style heuristic outperforms the dynamic programming based heuristic in both cases however, it gives poor accuracy as compared to dynamic programming based heuristic as shown in Fig. 4d. Moreover, both algorithms significantly reduces the runtime of optimization problem as compared to exact solution. The running time of the algorithm can be further reduced by using powerful computing servers and optimized programming implementation.

Furthermore, we also compare the average runtime of proposed heuristic algorithm for large input instances and storage technologies. Fig. 5a and 5b plots the CPU time consumed by greedy style heuristic for deciding the storage technologies for multiple IoT application data types. It can be seen that as the number of service requirements and IoT application data types increases, the runtime of greedy style heuristic increases. This is because for large number of IoT data types with many service requirements require the algorithm to search through multiple storage technologies to find the storage solution. Fig. 5c shows the runtime of dynamic programming based heuristic for multiple IoT data types. It can be seen that for large number of data types and storage technologies, the runtime of algorithm increases.

5) Sensitivity Analysis: We perform the sensitivity analysis to investigate the worst case performance of the heuristic algorithm for high workload. Fig. 6a shows the accuracy comparison of the proposed heuristics respectively for different IoT application data types, for cloud, storj and multi-cloud storage technologies. It can be seen that increasing the number of IoT application doesn't affect the accuracy achieved by the proposed algorithms. This is because of the fact that the storage decision is taken individually for each IoT application data types. However, increasing the number of IoT applications increases the runtime of the proposed algorithms. Similarly, Fig. 6b and 6c shows the cost comparison and accuracy of the proposed heuristics respectively for different number of storage technologies for randomly generated IoT application requirements. These storage technologies parameters are generated randomly above and below the range of real world parameters of cloud, storj and multi-cloud. The difference in the optimization cost of the proposed algorithms with baseline solution can be seen in Fig. 6b. This is because of the fact that the increasing the number of storage technologies increases the chance of selecting a sub-optimal storage solution with high cost, while fulfilling the IoT application requirements. Similarly, we can see the dip in the accuracy of proposed heuristics for large number of storage technologies.

As one can see, the accuracy of DP-based heuristic algorithm always outperforms all the types of GS-based heuristics even for the large number of input IoT applications and storage technologies. However, it comes at the cost of increased computation time to select the storage technologies for IoT applications. The computation time of DP-based heuristic increase almost twice as compared to GS heuristic with 5 service requirements however, it is evident from the evaluation that DP-based heuristic always achieves better accuracy than GS heuristic algorithms in all settings.

C. Smart Contract Cost Analysis:

We also provides the gas cost estimation of the smart contract functions used to provide blockchain functionality in the adaptive middleware. As explained before, two main write functions are included in smart contract, which are used by fog node and middleware to interact with blockchain network. Fog node sends the hash of the IoT data as well as application ID to the smart contract function **storeHash** to them to the blockchain network while, the middleware stores the storage decision along with application ID and data ID to the blockchain network through smart contract function **storeDecision**. Furthermore, in order to migrate data from one storage solution to another, middleware records the migration decision in the blockchain network through a smart contract function **migration**. The storeHash function is called by the fog node, while the middleware calls the storeDecision as well as migration function.The gas cost estimation for these functions is given in Table. II. It can be seen that the average confirmation time for a transaction is 42.6 seconds. However, assuming that the decision optimization problem will be solved once a day, consequently a single decision transaction will be sent to the blockchain network, 42.6 seconds is still a very small amount of time compared to 24 hours. At the time of writing this paper, the gas price and average confirmation time was taken from https://ethgasstation.info/.

IX. Conclusion

In this work, we introduce BlockAM: an adaptive middleware for the selection of storage technology for a specific IoT application based on the service requirements. The proposed middleware performs continuous monitoring of the storage solutions as well as IoT application parameters and adapts the best possible storage solution for specific IoT application. We model the cost-minimization storage selection problem and propose two polynomial-time heuristic algorithms to find the data storage solution based on IoT application's service requirements. We also employ blockchain to store IoT data on-chain as well as to ensure integrity, auditability and accountability in middleware architecture. Results show that the DP heuristic and GS heuristic achieves up to 92% and 80% accuracy respectively. Moreover, the price associated with a specific IoT application data storage decrease by up to 31.2% by employing our middleware solution.

References

[1] MB Mollah and et al. Secure data sharing and searching at the edge of cloud-assisted internet of things. *IEEE Cloud Computing*, pages 34–42, 2017.

[2] Ruinian Li and et al. Blockchain for large-scale internet of things data storage and protection. *IEEE Trans on Services Computing*, 2018.

[3] Qian Wang and et al. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proc. of ESORICS'09,*, pages 355–370. Springer, 2009.

[4] R Sravan Kumar and Ashutosh Saxena. Data integrity proofs in cloud storage. In *Proc. of COMSNETS'11*, pages 1–4. IEEE, 2011.

[5] Ryan K. L. Ko, Bu-Sung Lee, and Siani Pearson. Towards achieving accountability, auditability and trust in cloud computing. In *ACC*, 2011.

[6] Kaiwen Zhang and et al. Deconstructing blockchains: Concepts, systems, and insights. In *DEBS*, pages 187–190, 2018.

[7] Joseph R and et al. Blockchain-based distributed cloud storage digital forensics: Where's the beef? *IEEE Security & Privacy*, 17(1), 2019.

[8] David Vorick and Luke Champine. Sia: Simple decentralized storage. *White paper available at https://sia. tech/sia. pdf*, 2014.

[9] Shawn Wilkinson and et al. Storj: A peer-to-peer cloud storage network v2.0. https://storj.io/storjv2.pdf, 2016.

[10] Filecoin. [Online; accessed September 20, 2017].

[11] Nick L and et al. Safecoin: The decentralised network token, 2015.

[12] Kaiwen Zhang and et al. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *ICDCS*, pages 1337–1346, 2018.

[13] De Capitani DV and et al. Supporting application requirements in cloud-based iot information processing. In *IoTBD'16*, pages 65–72, 2016.

[14] Ansar Rafique and et al. Towards an adaptive middleware for efficient multi-cloud data storage. In *CrossCloud'17*, page 4. ACM, 2017.

[15] Zhe Wu and et al. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proc. of ACM Symposium on Operating Systems Principles*, pages 292–308. ACM, 2013.

[16] Mohammed A AlZain and et al. Mcdb: using multi-clouds to ensure security in cloud computing. In *In DASC'11*, pages 784–791, 2011.

[17] Dan Dobre and et al. Hybris: Robust hybrid cloud storage. In *Proc. of ACM Symposium on Cloud Computing*, pages 1–14, 2014.

[18] Christian Cachin, Robert Haas, and Marko Vukolic. Dependable storage in the intercloud. *IBM research*, 3783:1–6, 2010.

[19] Kevin D Bowers and et al. Hail: A high-availability and integrity layer for cloud storage. In *Proc. of the 16th ACM conference on Computer and communications security*, pages 187–198, 2009.

[20] Hossein Shafagh and et al. Towards blockchain-based auditable storage and sharing of iot data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, pages 45–50. ACM, 2017.

[21] Ali Dorri and et al. Towards an optimized blockchain for iot. In *Proc. of the second IoTDI'17*, pages 173–178. ACM, 2017.

[22] Kolbeinn Karlsson and et al. Vegvisir: A partition-tolerant blockchain for the internet-of-things. In *IEEE ICDCS'18*, pages 1150–1158, 2018.

[23] Ansar Rafique and et al. Scope: self-adaptive and policy-based data management middleware for federated clouds. *Journal of Internet Services and Applications*, 10(1):2, 2019.

[24] Abu-Libdehand et al. Racs: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 229–240. ACM, 2010.

[25] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[26] Syed Muhammad Danish. A blockchain-based adaptive middleware for large scale internet of things data storage selection. In *Proc. of Middleware'19 Doctoral Symposium*, pages 17–19, 2019.

[27] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8. IEEE, 2007.

[28] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[29] Eli Ben Sasson and et al. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[30] Bhaskar P R and et al. A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51. Ieee, 2009.

[31] Ranjita B and et al. Understanding availability. In *International Workshop on Peer-to-Peer Systems*, pages 256–267. Springer, 2003.

[32] Ivan B and et al. Approximation algorithms for data placement problems. *SIAM Journal on Computing*, 38(4):1411–1429, 2008.

[33] Jiangtao Z and et al. Data centers selection for moving geo-distributed big data to cloud. *Journal of Internet Technology*, 20(1):111–122, 2019.

[34] Nevena L and et al. Solving the uncapacitated facility location problem using message passing algorithms. In *AISTATS'10*, pages 429–436, 2010.

[35] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of algorithms*, 31(1):228–248, 1999.

[36] Bin G and et al. Winning at the starting line: Joint network selection and service placement for mobile edge computing. In *IEEE INFOCOM'19*, pages 1459–1467, 2019.

[37] Thomas HC and et al. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.