engineering proceedings

MDPI

Proceedings

A Proximal Algorithm for Fork-Choice in Distributed Ledger Technology for Context-Based Clustering on Edge Computing [†]

Rahim Rahmani *, Ramin Firouzi and Mahbub Alam

Department of Computer and Systems Sciences, Stockholm University, 16407 Stockholm, Sweden; ramin@dsv.su.se (R.F.); mahbub@dsv.su.se (M.A.)

* Correspondence: rahim@dsv.su.se

† Presented at the 7th Electronic Conference on Sensors and Applications, 15–30 November 2020; Available online: https://ecsa-7.sciforum.net/.

Published: 15 November 2020

Abstract: The major challenges of operating data-intensive of Distributed Ledger Technology (DLT) are (1) to reach consensus on the main chain as a set of validators cast public votes to decide on which blocks to finalize and (2) scalability on how to increase the number of chains which will be running in parallel. In this paper, we introduce a new proximal algorithm that scales DLT in a large-scale Internet of Things (IoT) devices network. We discuss how the algorithm benefits the integrating DLT in IoT by using edge computing technology, taking the scalability and heterogeneous capability of IoT devices into consideration. IoT devices are clustered dynamically into groups based on proximity context information. A cluster head is used to bridge the IoT devices with the DLT network where a smart contract is deployed. In this way, the security of the IoT is improved and the scalability and latency are solved. We elaborate on our mechanism and discuss issues that should be considered and implemented when using the proposed algorithm, we even show how it behaves with varying parameters like latency or when clustering.

Keywords: distributed ledger technology; blockchain; Internet of Things (IoT); edge computing; IoT security

1. Introduction

Distributed Ledger Technology (DLT) [1] is a key technology that manages and controls nodes in peer-to-peer applications that are mainly focused on payments and on record keeping in blockchain networks. This allows Internet of Things (IoT) applications and edge computing systems to operate at any point and the IoT application will be able to incorporate heterogeneous context entities in the heterogeneous domain, this heterogeneity mandates looking into interoperability reasoning approaches. The existing problems of using DLT for large scale distributed networks such as the IoT are highlighted by multiple different technologies trying to solve scalability issues while suffering from growing pains. Regardless of the popularity of the blockchain architecture, most implementations are limited regarding its practicality. These limitations include the number of transactions per second (throughput), the size of each block, the size of the chain, and the size of electronic signatures [2]. In the early implementations of blockchains, these limitations were not seen as issues but emerged later with the increasing number of nodes in the networks. To summarize the current situation of blockchains: they cannot scale the number of concurrent transactions with the users they likely are going to have [2]. There are some approaches suggested to run multiple blockchains simultaneously for scaling blockchain networks by increasing the number of chains or the so-called shards running in parallel but the scalability of the network will increase linearly. Those approaches are vulnerable to attack and identity/signature management challenges. The second drawback of some of the approaches is relying on proof of work (PoW) to reach a consensus, which is slow and energy inefficient.

In this paper, we propose a simple method to obtain parallel and distributed context proximal for reaching consensus and scalability by context-based clustering. Our contribution is a consensus protocol that requires no PoW mining and the method is based on the Alternative Direction Method Multiplier (ADDM) [3] algorithm and the key is to split DLT to provide a feeless and scalable DLT network to support data and value transfer.

2. Related Work

A distributed ledger is a database that is shared and synchronized across network nodes consensually. Participants at each node of the network can access the recordings shared across the nodes in the network and can own an identical copy of them. DLT provides a certain level of security guarantees to the data recorded in the ledger. Therefore, DLT is being used in different IoT areas, such as intelligent transportation [4–6], Industry 4.0 [7], healthcare [8,9] smart grid [10,11], and supply chain management [12], etc. For example, to preserve the privacy of patients, [9] introduced an attribute-based signature scheme, using blockchain technology. The approach in [6] proposes a blockchain ecosystem model for electric vehicle and charging pile management, which uses elliptic curve cryptography to calculate hash functions of electric vehicles and charging piles. In the following sections, we will introduce analyzing work for solving scalability and consensus (both global and general) issues raised by integrating DLT with IoT work regarding Edge computing, which is closely related to the mechanism proposed in this paper.

2.1. Context-Based Proximity for Scalability

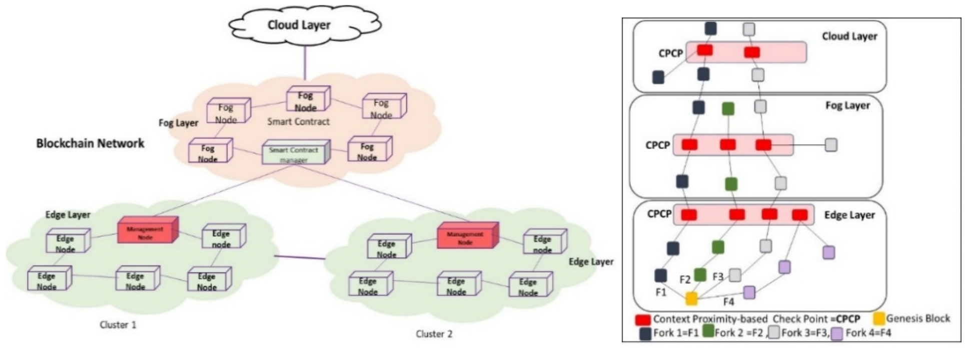
Among these works, [13] proposed a distributed IoT network architecture called DistBlockNet. DistBlockNet realized scalability and flexibility with the help of blockchain by not using a central controller. In DistBlockNet, all controllers in the IoT network are interconnected in a distributed blockchain network, in this way, each IoT forwarding device in the network can easily and efficiently communicate. Compared with DistBlockNet, the IoT devices in our mechanism can join the blockchain network depending on their capability. In addition, context information of the IoT devices is also considered during the selection of the head of the clusters. Ethereum is also aiming to introduce a new technology called Sharding, which was made to fight the problem of scaling [14]. Sharding is Ethereum's solution to one of the scalability problems of DLTs, the fact that every node needs to store a full, up-to-date copy of the blockchain, which is always growing. This places big storage requirements on nodes as well as slowing down transaction speeds of the network. By splitting the blockchain state and history into different shards and delegating them to different nodes, it is possible to reduce the amount of storage needed in each node [14]. According to the Ethereum development team, this also makes it possible for certain nodes to only verify transactions regarding a certain shard instead of the whole blockchain, which in turn allows different nodes to verify transaction blocks in parallel, speeding up the rate of transactions. The DLT context-based approaches using sharding will not be able to limit the requirements of PoW and Proof of Stake (PoS) mining to reach consensus.

2.2. Fog/Edge Computing for DLT Integrating with the IoT

Fog/Edge computing has been widely used in the IoT. Billions of IoT devices upload their data to the cloud, Fog, or Edge through global or local Internet, and the data are processed and used by using virtualization technology. In the context of integrating DLT with the IoT, several works have also been done [15–17]. IOTA [15] proposed a blockchain network using the Directed Acyclic Graph (DAG) as the ledger data structure to organize transaction data on every IOTA node. IOTA is used for IoT applications such as in [18] it was used as a marketplace where electricity trading is directly done by IoT devices as sellers or buyers on the Edge part of the access network. Compared with these works, our work uses the layered architecture to introduce clustering of IoT devices dynamically into groups based on proximity context information.

3. Background and Motivation

In this section, we briefly discuss the background and our motivation behind this work. For the reason of scalability, IoT devices in a cloud-based environment are normally supported in a hierarchical structure consisting of different layers of equipment performing the computation functions. In our work, we consider two layers of computations. The Fog layer is the high layer with distributed computations and the Edge layer is the low layer of distributed computations. The IoT devices are connected to the Edge layer equipment as shown in Figure 1.



(**a**) (**b**)

Figure 1. Layered distributed computation architecture supporting DLT for IoT devices (**a**) To support DLT for IoT applications in a scalable way. (**b**) Illustration of a chain block of transactions with four forks. The bottom (yellow) node is the genesis block. Red nods represent the finalized Context Proximity Check Point.

Figure 1a illustrates an example of layered distributed architecture for IoT devices supporting DLT and will address (1) the Edge computing with DLT (2) DLT for clustered IoT devices. Figure 1b shows context proximity-based DLT consensus indicating the approval of a single transaction and its fork-choices.

4. Methods

4.1. *System Model of the Proximity Consensus Algorithm*

In order to address scalability and be different from existing approaches that are using sharding [14] to improve scalability, sharding splits the blockchain network into partitions in a way that no all nodes need to process all incoming transactions. We propose logical proximity spatial distributions in the DLT-Based IoT-Edge as shown in Figure 1. Our approach limits transaction forwarding and does not allow any inconsistency in the smart contract changes. As mentioned in the previous section, a set of IoT devices will be managed as a vertical spatial distribution based on context proximity. The context in sensor clustering has been discussed before, but in all cases, the definition of context is specific. Moreover, their solution is limited to neighboring nodes. To the best of our knowledge, the concept of logical clustering of nodes based on context is new and [19,20] was the first attempt of the proposed concept. This new concept will allow resources (data, services) to be shared among different spatial distributed nodes and they can share resources through distributed collaboration. Once the clustering is done, then each cluster is identified through a context-ID, which is defined based on context similarity and published on the Internet. In this study, we extended the context-ID [19,20] to be a relational proximity spatial distribution by assigning different weights to the connections between nodes to reduce traffic on critical links and besides scalability, the algorithm also allows different context-based parts to run different combinations of consensus, ledger, and transaction models. Furthermore, dynamically clustering can easily inter-operate between clusters.

Context proximity of large-scale spatial distributions of IoT devices and the population of wireless sensors networks (WSN) are discussed in the next section.

Context Proximity Spatial Distributions

A Distributed Ledger can serve as distributed control in Fog and Edge nodes, being able to negotiate distributed context-based coordination problems to automated DLT for context-based vertical-distribution analysis by generating in the population of WSN and developing the scenarios for simulation to inflate the algorithm for co-aggregation analysis of clustered IoT devices at the Edge. The approach to context proximity provided direct support for such a model, which could enable users/validators to acquire validations of proximity over spatial distribution thus enabling consensus and sharing. We deployed the Alternative Direction Method Multiplier (ADMM) [3] based method to split the genesis block to fork blocks as shown in Figure 1b. The separability of the forks gives the algorithm the ability to evaluate the proximal operation of the forks to operate in parallel and distributed.

4.2. Consensus by Sharing within Clustering

The consensus mechanism allows an effective majority of maintainers to decide in the case of DLT what transactions to include in their ledgers. The majority of maintainers can be expressed in consensus form by introducing a sequence of decision blocks with included transactions. We interpreted consensus ADMM based as a method for solving problems in which the blocks and transactions are distributed across multiple ledgers. Each ledger only has to handle its own blocks and transactions. The clustering stage involves clustering the nodes that represent a simulated distributed system and evaluating and comparing the produced clustering to other clustering results to obtain the best result possible. The ranking and leader election or cluster head stage involves ranking the nodes in the simulated distributed system, which is done by ranking them according to their distance to the medoid of the best clustering produced in the clustering stage. Then, simple leader election or cluster head functions are performed using the ranks as input. In the last stage, program synthesis, the ranks of the nodes that were produced in the previous stage and their respective leaders are used as input-output examples to synthesize a program that is consistent with the provided examples. In the sub-activity, Assess and Select, different machine learning techniques were investigated for use in the clustering. At one point, reinforcement learning was considered for the program synthesis stage of the artifact. We used DeepCoder's [21,22] default implementation of program synthesis because of time constraints as well as concerns whether reinforcement learning was appropriate for the problem to be solved. Something that also contributed to this decision was that the reinforcement learning algorithms that could be used for this purpose were deemed to be more complicated than DeepCoder's default implementation.

4.2.1. Context-Based Clustering

Clustering was done as a pre-processing step to generate clusters based on context values for nodes in a dataset, simulating a distributed system. The algorithm used for clustering is the k-medoids algorithm as implemented in Python by the pyclustering library [23] (which in turn is an implementation of PAM), while the *k-means*++ implementation in the pyclustering library was used to choose the initial points. The *k-medoids* algorithm was used because it is more robust to outliers than k-means and because using an actual data point (medoid) as the representative object of a cluster facilitates leader election in later stages. There is also no risk that the choice of initial points will produce incorrect clusters, even though that risk would have been eliminated by the use of *k-means*++ as an initializer in any case. *K-means*++ was used as an initializer to ensure a fair distribution of the initial points and because it can improve the speed and accuracy of the clustering according to [24]. To determine the best clustering for a certain dataset, the *k-medoids* algorithm was run with different values of *k* and each produced clustering was evaluated based on its average Silhouette value. The average Silhouette value was calculated by a function called *sklearn.metrics.**silhouette_score*** from the scikit-learn module [25] in Python. Scikit-learn is a machine learning module "for medium-scale supervised and unsupervised problems [26]. Clustering was done on a two-dimensional dataset of simulated context values, although it is fully possible to perform clustering on a high-dimensional dataset. There are even algorithms such as subspace clustering that seek to find clusters in different subspaces within a dataset [27]. However, we chose to instead put the complexity in the generation of context values that may be based on an aggregation of many different values. The motivation for this decision is that clustering high-dimensional data is very computationally expensive and would make it harder to meet the performance requirements imposed on the concept. The clusters were then passed to the next stage for ranking and leader election or the cluster head to be performed.

4.2.2. Ranking and Cluster Head or Leader Election

The cluster head or leader election algorithm that was to be implemented in this paper is a variant of the monarchical leader election algorithm (Algorithm 2.6) [28]. The leader election algorithm requires each process to have a rank associated with it, which was done in a ranking function that assigns a rank to each node based on its distance to the representative object of its cluster, the medoid. The distance metric that was used was the Manhattan distance. The ranks were then passed to the leader election function which elects the highest-ranking node as the leader of each cluster, for use as an oracle later on. The ranks were then passed to DeepCoder in the next stage as the input examples and their medoids, or leaders, as the output examples, thus generating the input-output examples needed for DeepCoder to synthesize a program.

The algorithm defined in (Algorithm 2.6) [29] only allows the election of a new leader if the current leader has crashed, meaning it must be modified to allow the election of a new leader in other circumstances as well, for example when new nodes join the system. This was not done in this thesis because the system is simulated, and a perfect failure detector is assumed but would have to be done in a real system.

4.2.3. Program Synthesis

A program synthesis consistent with the input-output examples produced by the previous stages must be synthesized, or in other words, a hierarchical leader election program must be synthesized. This was done by using DeepCoder's LIPS [21,22] framework. No reference implementation of DeepCoder seemed to be available, but several third-party implementations could be found on GitHub. Since Python is the language used in our simulation, we chose to use the Python 3 implementation of DeepCoder [29]. It required Python 3 as well as machine learning and mathematical libraries such as Keras [30], Tensorflow [31], NumPy [32], and pandas [33]. Keras is used as a high-level interface that runs on a Tensorflow backend to facilitate the neural network while NumPy and pandas are widely used Python libraries for scientific computing and data manipulation and analysis.

5. Results and Performance Evaluation

In this section, we present the conceptual validation and performance evaluation of our proposed model with a concise and precise description of our experimental results. The validation and performance measurements are performed based on five different scenarios. All datasets were randomly generated by using the Python function random. **uniform** [34] or manually typed in at random. Some of the datasets were manually typed in to create natural clusters in the data to illustrate a more realistic scenario, rather than just a truly random uniform distribution. The randomly generated datasets are first and foremost used to test the scalability of the concept. These datasets represent the simulated context information for different nodes in a distributed system as floating-point numbers. They contain no duplicates but are not necessarily disjoint from each other as some parts of the smaller datasets are also subsets of the larger datasets (Scenario 2 includes some of the data in Scenario 1, Scenario 3 includes some of the data in Scenario 2, and Scenario 5 includes all the data from Scenario 4). All performance measurements were taken with the Python 3 built-in function time.**perf_counter** [35] because it uses the highest available clock resolution. For comparison, the elapsed real-time (commonly called wall time) was also measured with the function time.**time** [36] for the same tasks. All plots were created with Matplotlib. To get the best clustering available for each scenario, clustering was run for $k = 2$ to 10, for example, nine times.

5.1. K-Medoids Clustering

In Scenario 1, 29 nodes are present in a distributed system. The dataset was manually typed in at random. A situation that could correspond to a real-world example of this scenario is that one or several of the previous leaders have crashed or experienced link failures. Subsequently, new leaders must be elected. The best clustering that was found for Scenario 1 had an average Silhouette of 0.8485671038005586 and was achieved with $k = 2$. The best available clustering for Scenario 1 is thus two clusters as shown in Figure 2a. In the Scenario 2 clustering scenario, five new nodes are added to the system, increasing the total number of nodes to 34. The dataset was manually typed in at random. A situation that could be a real-world example of this scenario is that new nodes have been added to the system and the nodes must be clustered according to context for the system to assign a leader to each group of nodes. The best available clustering for Scenario 2 is three clusters as the best clustering found had an average Silhouette of 0.5863200762086528 and was achieved with $k = 3$. The best clustering for Scenario 2 is shown below in Figure 2b. The Scenario 3 clustering is meant to test the scalability of the artifact by adding 12 more nodes to the system. The dataset for Scenario 3 was also manually typed in at random. For Scenario 3, the best clustering had an average Silhouette of 0.8332508360470124 and was achieved with $k = 2$. Thus, the best available clustering for Scenario 3 is two clusters as shown in Figure 2c. The Scenario 4 clustering operates on a randomly generated dataset of 200 nodes in order to further test its generalization ability and scalability. The best available clustering for Scenario 4 is four clusters as shown below in Figure 2d, as the best Silhouette value of 0.3830742916987218 was achieved with $k = 4$. The Scenario 5 and final clustering scenario aims to further test the scalability of the system by adding 800 nodes to the system, for a total of 1000 nodes. The dataset was generated the same way as in Scenario 4. The Silhouette of the best clustering for Scenario 5 was 0.363725300745691 which was achieved with $k = 7$. The best available clustering for Scenario 5 is seven clusters as shown in Figure 2e. Silhouette analysis of *K*-medoids clustering on Scenario 5 sample data with $k = 10$ is shown in Figure 2f.

5.2. Inductive Program Synthesis (IPS) and Cluster Head or Leader Election

After all the clustering scenarios had finished running, the best clustering was used as the representative measurement of that scenario. All the representative clusters were then passed to DeepCoder for program synthesis. The input-output examples for DeepCoder were generated by ranking all the nodes in each cluster according to the distance to the medoid of the cluster. The medoid had the highest rank, the object closest to the medoid the next highest rank. Inductive program synthesis is capable of synthesizing a program consistent with the input-output examples for a leader election algorithm, and if so, how. Table 1 shows that DeepCoder is in fact able to synthesize a correct and consistent leader election program for the simple hierarchical leader election algorithm used in this paper. Both Depth-first search (DFS) and sort and add search techniques produced the same program, which led to increased confidence in the results. The synthesized program is of the format INPUT|FUNCTION, ARGUMENT meaning that a list is taken as the input to the program, which consists of the function MAXIMUM with variable 0 as the argument. As there is only one variable present in this program, the input, this is argument variable 0, and the function is applied to the input list. The steps used are how many steps the search technique had to take before finding a consistent program. Table 2 shows, the performance measurements for synthesizing a consistent program for the two different search techniques.

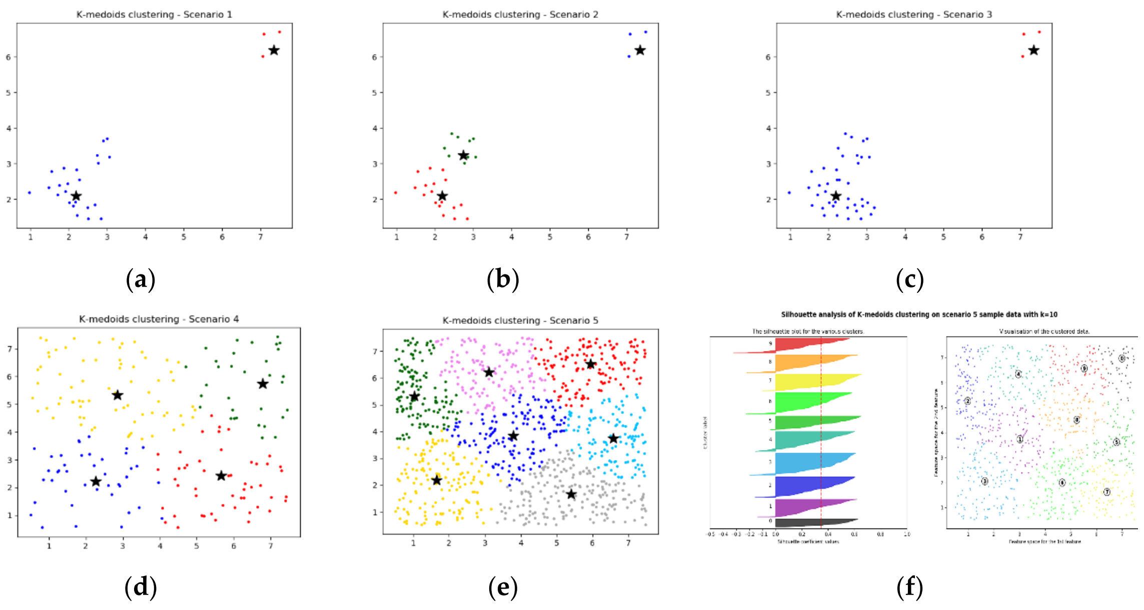


Figure 2. (**a**) Scenario 1 (**b**) Scenario 2 (**c**) Scenario 3 (**d**) Scenario 4 (**e**) Scenario 5 (**f**) Silhouette analysis of Scenario 5 with $k = 10$.

Table 1. Results for Inductive Program Synthesis (IPS).

Labels	Synthesized Program	Program Consistent	Steps Used
DFS	LIST|MAXIMUM,0	Yes	5
Sort and add	LIST|MAXIMUM,0	Yes	[2–5]

Table 2. Performance measurements for IPS.

Labels	DFS Run Time	DFS Wall Time	Sort and Add Run Time	Sort and Add Wall Time
Minimum	0.874 ms	0.0 ms	1.263 ms	0.997 ms
Maximum	0.964 ms	0.997 ms	1.442 ms	1.995 ms
Mean	0.888 ms	0.764 ms	1.315 ms	1.462 ms

6. Conclusions

The huge deployment of DLT within IoT infrastructure is an ongoing work. In this paper, we have proposed a novel proximal algorithm for fork-choice in DLT for context-based clustering on Edge computing. The algorithm was evaluated to determine how the cluster head can be used to bridge the IoT devices with the DLT network where a smart contract will be deployed. The verification results in Figure 2 show the algorithm ability and it can indeed be used to solve the practical problem of clustering nodes in IoT networks by context and cluster head or electing leaders for the produced clusters using IPS. It fulfills the requirements set upon it well, which brings the possibility of implementation on limited hardware. In a real system, the clustering stage of the algorithm should be implemented on a higher level than every node, for example, on Edge nodes in an IoT environment. The IPS stage is intended to be implemented on every node to facilitate leader election and the results make this possible.

References

1. Consocenti, M.; Vetro, A.; De Martin, J.C. Blockchain for the Internet of Things: A Systematic Litteratur review. In Proceedings of the IEEE/ACS 13th International Conference on Computer Systems and Applications, Agadir, Morocco, 29 November–2 December 2016.
2. Kim, S.; Kwon, Y.; Cho, S. A Survey of Scalability Solutions on Blockchain. In Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea South, 17–19 October 2018; pp. 1204–1207.
3. Bertsekas, D.P.; Tsitsiklis, J.N. Parallel and Distributed Computation: Numerical Methods; Prentice Hall: Upper Saddle River, NJ, USA, 1989.
4. Dorri, A.; Steger, M.; Kanhere, S.S.; Jurdak, R. BlockChain: A Distributed Solution to Automotive Security and Privacy. *IEEE Commun. Mag.* **2017**, *55*, 119–125.
5. Li, L.; Liu, J.; Cheng, L.; Qiu, S.; Wang, W.; Zhang, X.; Zhang, Z. CreditCoin: A Privacy-Preserving Blockchain-Based Incentive Announcement Network for Communications of Smart Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 2204–2220.
6. Huang, X.; Xu, C.; Wang, P.; Liu, H. LNSC: A security model for electric vehicle and charging pile management based on blockchain ecosystem. *IEEE Access* **2018**, *6*, 13565–13574.
7. Li, Z.; Kang, J.; Yu, R.; Ye, D.; Deng, Q.; Zhang, Y. Consortium Blockchain for Secure Energy Trading in Industrial Internet of Things. *IEEE Trans. Ind. Informa.* **2017**, *14*, 3690–3700.
8. Esposito, C.; de Santis, A.; Tortora, G.; Chang, H.; Choo, K.-K.R. Blockchain: A Panacea for Healthcare Cloud-Based Data Security and Privacy? *IEEE Cloud Comput.* **2018**, *5*, 31–37.
9. Guo, R.; Shi, H.; Zhao, Q.; Zheng, D. Secure Attribute-Based Signature Scheme with Multiple Authorities for Blockchain in Electronic Health Records Systems. *IEEE Access* **2018**, *6*, 11676–11686.
10. Gao, J.; Asamoah, K.O.; Sifah, E.B.; Smahi, A.; Xia, Q.; Xia, H.; Zhang, X.; Dong, G. GridMonitoring: Secured sovereign blockchain based monitoring on smart grid. *IEEE Access* **2018**, *6*, 9917–9925.
11. Liang, G.; Weller, S.R.; Luo, F.; Zhao, J.; Dong, Z.Y. Distributed Blockchain-Based Data Protection Framework for Modern Power Systems against Cyber Attacks. *IEEE Trans. Smart Grid* **2018**, *10*, 3162–3173.
12. Tian, F. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In Proceedings of the 2016 13th International Conference on Service Systems and Service Management (ICSSSM), Kunming, China, 24–26 June 2016.
13. Sharma, P.K.; Singh, S.; Jeong, Y.-S.; Park, J.H. DistBlockNet: A Distributed Blockchains-Based Secure SDN Architecture for IoT Networks. *IEEE Commun. Mag.* **2017**, *55*, 78–85.
14. Sharding FAQs. Available online: https://github.com/ethereum/wiki/wiki/Sharding-FAQs (accessed on 10 October 2020).
15. IOTA. Available online: https://www.iota.org/ (accessed on 10 October 2020).
16. Xia, Q.; Sifah, E.B.; Asamoah, K.O.; Gao, J.; Du, X.; Guizani, M. MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain. *IEEE Access* **2017**, *5*, 14757–14767.
17. Sharma, P.K.; Chen, M.-Y.; Park, J.H. A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT. *IEEE Access* **2018**, *6*, 115–124.
18. Parker, J.; Chitchyan, R.; Angelopoulou, A.; Murkin, J. A block-free distributed ledger for p2p2 energy trading: Case with iota? In *International Conference on Advanced Information Systems Engineering*; Springer: Cham, Switzerland, 2019; pp. 111–125.
19. Rahmani, R.; Rahman, H.; Kanter, T. Context-Based Logical Clustering of Flow-Sensors Exploiting HyperFlow and Hierarchical DHTs. In Proceedings of the 4th International Conference on Next Generation Information Technology, Noida, India, 26–27 September 2013.
20. Rahman, H.; Rahmani, R.; Kanter, T. Multi-modal context-aware reasoNer (CAN) at the edge of IoT. *Procedia Comput. Sci.* **2017**, *109*, 335–342.
21. Balog, M.; Gaunt, A.L.; Brockschmidt, M. DeepCoder: Learning to Write Programs. *Machine Learn.* **2017**. arXiv:1611.01989.
22. Deep-coder. Available online: https://github.com/HiroakiMikami/deep-coder (accessed on 5 September 2020).
23. Pyclustering library. Available online: https://github.com/annoviko/pyclustering (accessed on 6 September 2020).
24. Arthur, D.; Vassilvitskii, S. *k-means++: The Advantages of Careful Seeding*; Stanford InfoLab: Stanford, CA, USA, 2007.

25. Scikit-Learn Module. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn-metrics-silhouette-score (accessed on 6 September 2020).
26. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Vanderplas, J. Scikit-learn: Machine learning in Python. *J. Machine Learn. Res.* **2011**, *12*, 2825–2830.
27. Parsons, L.; Haque, E.; Liu, H. Subspace clustering for high dimensional data: A review. *Acm Sigkdd Explorations Newsletter* **2004**, *6*, 90–105.
28. Cachin, C.; Guerraoui, R.; Rodrigues, L. *Introduction to Reliable and Secure Distributed Programming*, 2nd ed.; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2011.
29. Pythone DeepCoder. Available online: https://github.com/dkamm/deepcoder (accessed on 5 September 2020).
30. Keras. Available online: https://keras.io/ (accessed on 5 September 2020).
31. Tensorflow. Available online: https://www.tensorflow.org/ (accessed on 5 September 2020).
32. Numpy. Available online: https://numpy.org/ (accessed on 5 September 2020).
33. Pandas. Available online: https://pandas.pydata.org/ (accessed on 10 September 2020).
34. Python Random Function. Available online: https://docs.python.org/3/library/random.html#random.uniform (accessed on 10 September 2020).
35. Python 3 Built-in Function time.perf_counter. Available online: https://docs.python.org/3/library/time.html#time.perf_counter (accessed on 10 September 2020).
36. Wall Time with the Function time.time. Available online: https://docs.python.org/3/library/time.html#time.time (accessed on 10 September 2020).