Rydzi, Filip; Truong, Linh

Sharing Blockchain Performance Knowledge for Edge Service Development

Published in:
Proceedings - 2019 IEEE 5th International Conference on Collaboration and Internet Computing, CIC 2019

DOI:
10.1109/CIC48465.2019.00012

Published: 01/01/2019

Document Version
Peer reviewed version

Please cite the original version:
Rydzi, F., & Truong, L. (2019). Sharing Blockchain Performance Knowledge for Edge Service Development. In *Proceedings - 2019 IEEE 5th International Conference on Collaboration and Internet Computing, CIC 2019* (pp. 20-29). [8998488] IEEE. https://doi.org/10.1109/CIC48465.2019.00012

Sharing Blockchain Performance Knowledge for Edge Service Development

Filip Rydzi
Independent, Slovakia
Email: rydzi.filip@gmail.com

Hong-Linh Truong
Department of Computer Science, Aalto University, Finland
Email: linh.truong@aalto.fi

***Abstract*—The integration of Internet of Things (IoT) and cloud services with edge technologies has enabled the development of many new types of edge services, which leverage blockchain features for cross-organizational, traceable and verifiable records. However, developing such edge services with blockchain features requires not only knowledge about complex blockchain technologies but also how blockchain technologies coexist with edge computing service models and architectures and deployments. In the context of edge service development, coupling edge systems, software models for edge services and blockchain technologies is complex. Thus, a strong collaboration and knowledge sharing for edge systems and blockchain technologies will help addressing many concerns of the developer. However, there is a lack of frameworks for sharing knowledge about blockchain software artefacts and deployments for edge services. In this paper, we present various types of information linking blockchain performance with service deployments at different levels. We represent and associate benchmarked performance information of blockchain operation and blockchain infrastructural services with common edge service interactions and resource deployments. Based on that, we develop a service offering blockchain knowledge to the developer seeking relevant blockchain operation information for their development decisions. We will present a prototype of our framework with benchmarked information obtained from experiments with Ethereum and Hyperledger.**

***Index Terms*—blockchain, edge computing, performance, knowledge sharing, software development, benchmark**

I. Introduction

Edge Services (ES) are complex, consisting of various IoT, edge and also cloud components[1], [2], [3], [4]. Blockchain-based features in ES environments have recently been intensively studied by researchers [5], [6], [7]. Due to the problem complexity, developers are faced with many challenges [5], [8], [9] in engineering blockchain-based applications in ES, for example, how to choose a suitable deployment of blockchain features into ES components [10]. Through our motivation discussion (see Section II), we show that there are many complex issues and various types of performance knowledge that the developer wants to obtain for designing and deciding blockchain features and suitable deployments. However, there is a lack of generic frameworks to help developers sharing and recommending such suitable deployments of blockchain features for blockchain-based applications in ES. Related work (see Section V) applies design patterns on blockchain systems as well ES deployment patterns. There are also many description languages [11] for specifying ES deployments. However, they miss models of knowledge about the blockchain deployment topologies and artefacts for blockchain features.

Our goal is to build a generic framework to manage the knowledge about blockchain performance for ES. Such knowledge can be shared among developers, facilitating the collaboration among them. A developer could provide benchmark information to our framework, helping to enrich the knowledge, while other developers can utilize the knowledge through searches and recommendations. To this end, at the center of our framework is a service for management of the knowledge. In the architecture of our framework (see Section III), performance knowledge can be gathered from existing benchmarks and monitoring tools for blockchain, such as [12], [13], and can be combined with deployment models and service operations.

To represent useful blockchain performance knowledge, we propose a comprehensive model (in Section III), which covers software topologies, composed of ES components, blockchain software artefacts deployed to the topologies, ES infrastructures and quality metrics measured. Having such a suitable model enables us to aggregate data across various deployments and build knowledge from the data. Based on that, we provide a service managing knowledge and recommending performance information to the developer. Our framework has been implemented into a prototype called GIAU (knowledGe for blockchaIn Applications and Utilities). In this paper, we will evaluate our prototype, which is open source in GitHub[1], through examples how the developers could benefit from mobile edge computing scenarios.

The remainder of this paper is structured as follows. Section II presents our motivation scenarios. In Section III we present a high-level architectural overview of GIAU and detail how we manage the knowledge. Prototype and experiments are given in Section IV. Related work is elaborated in Section V. Remaining issues of the work will be discussed in Section VI.

II. Motivation

Development of blockchain-based ES involves activities at multiple levels. At the highest level the developer may have to choose an underlying blockchain infrastructure for ES, e.g., to use Ethereum [14] or Hyperledger Fabric [15], [16]. At a lower level, a suitable deployment of blockchain features in

[1]https://github.com/rdsea/blockchainbenmarkservice/tree/master/giau

a topology of ES components has to be decided. Consider, for example, the developer develops a blockchain-based application for vehicle-to-everything (V2X) communication[2] that will be executed in an ES topology, composed of a vehicle, a smartphone, an edge device and an edge data center. The developer needs to decide which blockchain software artefacts should be deployed to which nodes of the topology. Such a decision on the deployment is very challenging for the developers. Another type of knowledge is at the level of blockchain nodes, e.g., the developer has to decide whether to deploy a Hyperledger Fabric peer node [15] to an edge device or to an edge data center. If s/he would prefer to deploy the peer node to the edge device, it might decrease latency, but increase the infrastructure cost, as opposed to a deployment in the edge data center.

At the lowest level, blockchain features are executed in blockchain nodes running in ES components; nodes carry out typical blockchain operations, such as mining, creating and verifying transaction, and accepting transactions. Thus, knowing characteristics of blockchain operations is crucial. To demonstrate the importance of understanding operations, let us consider a code excerpt[3] in Listing 1, which stores a warning message from a vehicle on a blockchain system (Hyperledger Fabric in this example). We assume the piece of code is executed in one of components of ES and `fabricClient` wraps a connection to a blockchain node, which is deployed to an ES component as well. The operation, which submits a transaction, containing the warning message, to blockchain, is invoked on line 20 in Listing 1.

Listing 1. Example of blockchain transaction submission

```
1  let request = {
2    chaincodeId: 'warn_cc',
3    fcn: 'createWarning',
4    args: [1, 'obstacle detected', latitude, longitude],
5    chainId: 'mychannel'
6  };
7  let tx_id = fabricClient.newTransactionID(false);
8  request.txId = tx_id;
9  // send the transaction proposal to the peers
10 channel.sendTransactionProposal(request).then((results) =>
       {
11   let proposalResponses = results[0];
12   let proposal = results[1];
13   if (proposalResponses && proposalResponses[0].response &&
           proposalResponses[0].response.status === 200) {
14     let txRequest: TransactionRequest = {
15       proposalResponses: proposalResponses,
16       proposal: proposal,
17       txId: tx_id
18     };
19     // submitting a transaction to blockchain
20     return channel.sendTransaction(txRequest);
21   } else {
22     throw new Error(`Transaction proposal is bad.`);
23   }
24 }).then( (results) => {
25   // work with results of the transaction
26 } );
```

The developer is interested in the following questions: how long does it take to perform the operation, how much resources are needed to execute the operation, and eventually what is the dependency between the underlying infrastructure and performance of the operation. The answers to those questions are dependent on the deployment. These answers are important for the developer as they help to improve qualities of the applications.

In this paper, we first determine what data do we need for the knowledge and propose a structure capturing the knowledge. Having the structure we work on techniques of managing and providing the knowledge under a service for facilitating collaboration among developers.

III. Knowledge for blockchain applications and utilities

A. Architectural overview

Figure 1 depicts a high-level architectural overview of GIAU, our framework enabling knowledge sharing for deployment of blockchain to ES. Via *Deployment Pattern Service* a developer can manage all deployment patterns stored in GIAU. A deployment pattern is a graph consisting of ES components, represented as graph nodes, while edges of the graph represent interactions among the ES components. *Infrastructure Service* manages metadata about compute resources and network configurations in an ES infrastructure. The resources are provided by an external resource provider. The responsibility of *Software Artefact Service* is to manage metadata of blockchain software artefacts. GIAU uses external repositories of the blockchain software artefacts so we store only metadata concerning those artefacts; the artefacts are stored at an external provider (e.g. DockerHub). The metadata provide enough information for developers to obtain and execute those artefacts. *Blockchain Benchmark DaaS* represents and manages benchmark data, stored in the GIAU but provided by existing benchmark tools. Benchmark data are arranged into experiments. An experiment specifies a case in which blockchain operations have been been benchmarked (or to be benchmarked) in a concrete setting. Each experiment has a topology and quality metrics measured when benchmarking the experiment. A topology of the experiment is a deployment pattern, such that software artefacts (from *Software Artefact Service*) are deployed to the ES components involved in the pattern. Each of ES components in an experiment's topology has an associated resource (from *Infrastructure Service*); in many cases a resource is used as a container for the ES component. The purpose of *Recommendation Service* is to utilize knowledge stored by GIAU to give recommendations about deployment of blockchain artefacts to ES components involved in a topology of blockchain-based application.

B. Linking blockchain knowledge with service and infrastructure knowledge

Figure 2 illustrates a model of data representing the knowledge in GIAU. We will explain them in the following.

[2]https://networks.nokia.com/products/vehicle-to-everything

[3]Extracted from https://github.com/rdsea/blockchainbenmarkservice/blob/master/emulators/v2x_communication/src/main/hypfab/HypFabVehicleDAO.ts

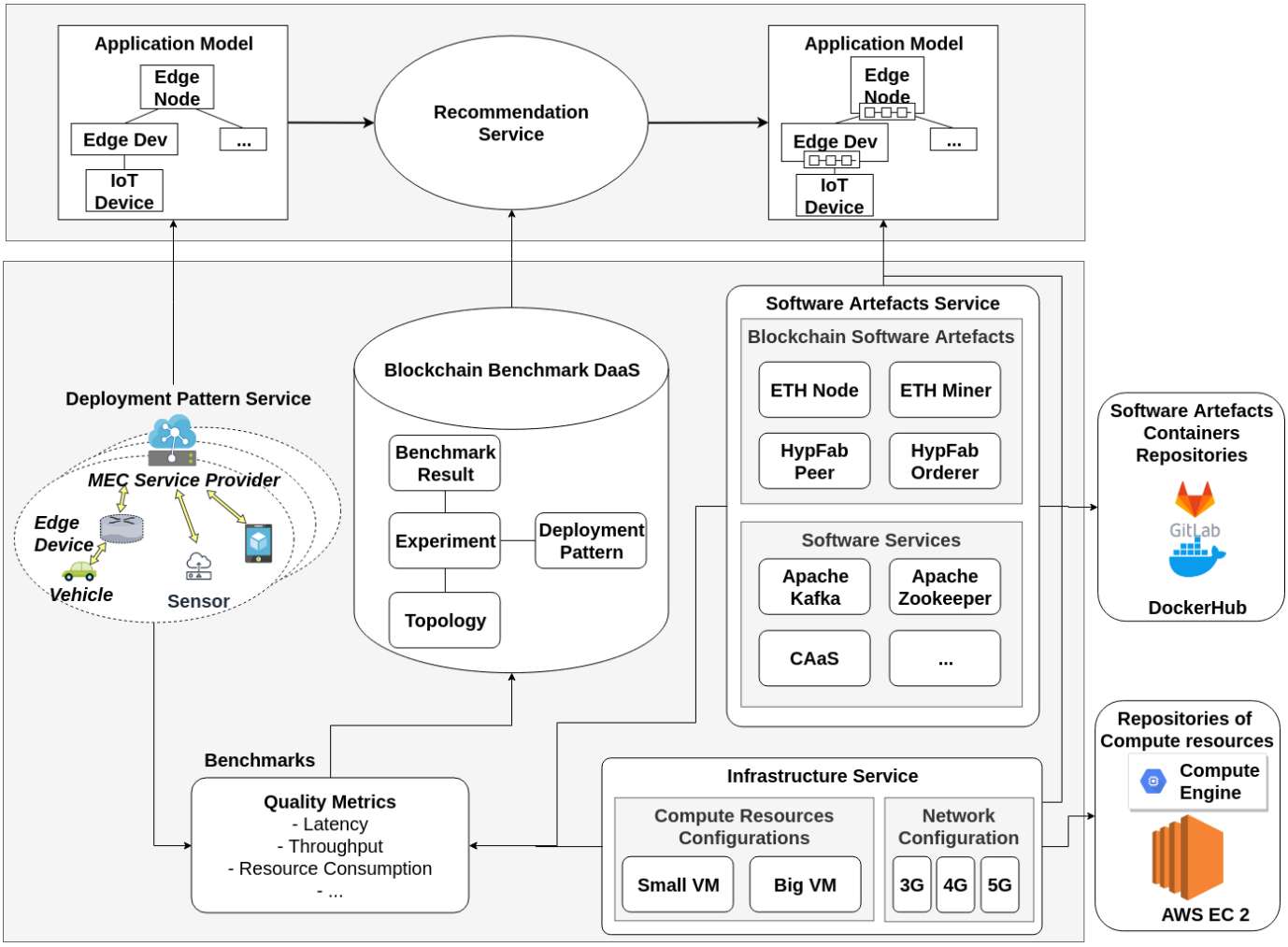

Fig. 1. High-level overview of GIAU framework

1) Capturing deployment patterns: Developer might be interested in patterns of blockchain interaction between ES components and how do those interactions perform concerning various quality metrics. Consider a following example: a developer is implementing an application, which utilizes blockchain for interaction between Internet of Things (IoT) devices and edge data-centers. S/he's is not sure if an edge device, added to the interaction between IoT devices and edge data-centers, would enrich performance of the interaction. To be able to answer that question, we have to consider the patterns of interaction and link them to benchmark results. The interaction patterns are represented via *DeploymentPattern* class. The *DPNode* class stands for an ES component participating in an interaction. In order to support various types of ES components, we create sub-classes of *DPNode*; they are: *DPCloud*, *DPEdgeNode*, *DPEdgeDev* and *DPThing*. *DPThing* represent IoT devices [10]. IoT devices are interacting among each other, with devices at the edge [10], with edge data centers and with cloud services. The devices at the edge are represented by the *DPEdgeDev* class, edge data-centers by the *DPEdgeNode*. *DPCloud* is used to capture cloud services. The model is extendable to support new ES components, which can be introduced by a new *DPNode*'s subclass. Each *DPNode* is identified by its *id*. Attribute *name* presents its caption. A *DPNode*'s *peers* are other *DPNodes*, which are connected to the *DPNode* by an edge in the graph representing a deployment pattern.

2) Representing blockchain software artefact information: We represent knowledge at two levels: blockchain nodes and blockchain operations. At the blockchain node level, a blockchain node is represented by an executable blockchain software artefact; examples of blockchain artefacts, as considered in this paper, are `Geth`[4], `Hyperledger Fabric peer node` [15], and `Hyperledger-Fabric orderer node` [15]. At the blockchain operation level, blockchain operations, used to implement blockchain features in ES, include creating a transaction, verifying a transaction, and mining and accepting a block, and are carried out by blockchain software artefacts (nodes). Let us consider a developer who uses Hyperledger-Fabric. Listing 2 illustrates an example of in-

[4]https://github.com/ethereum/go-ethereum/wiki/geth

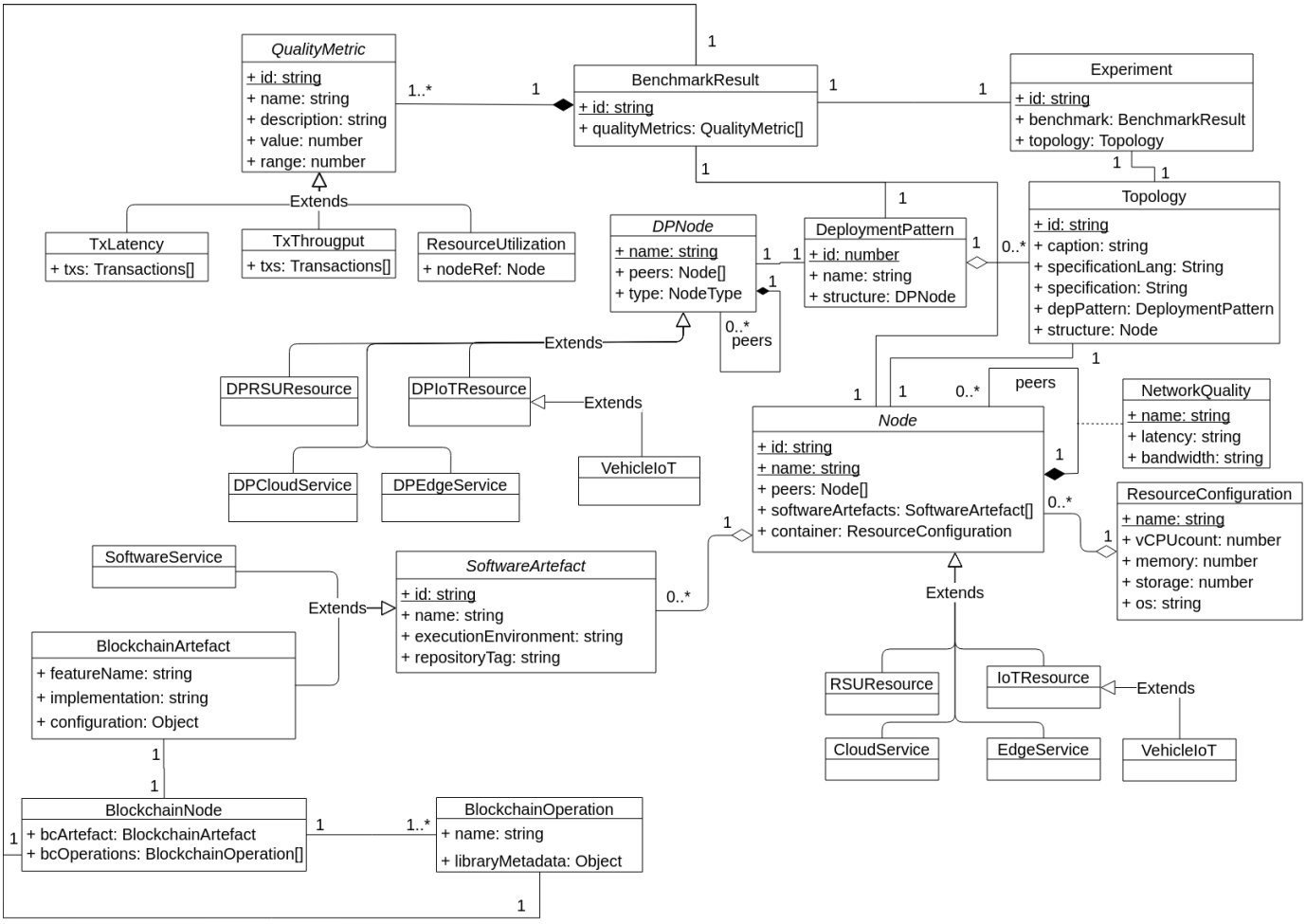

Fig. 2. Model of data stored in GIAU

formation representing knowledge at the level of a blockchain node whereas an example for blockchain operations is given in Listing 3, which can be carried out by the artefact in Listing 2. Both types of information can be extracted from existing benchmarks or software configurations in the developer work through manual actions and custom utilities. Some types of information are common for many developers, such as container images used and environment variables, but other specific information reflect the setting used by the developer.

As depicted in Figure 2 the knowledge is linked to the benchmarks results. Linking benchmark results to blockchain software artefacts information is important for giving recommendations (our approach for the recommendations is described in Section III-C). Blockchain artefact is represented via the class *BlockchainArtefact* depicted in Figure 2. *BlockchainArtefact* is a subclass of *SoftwareArtefact*. The artefacts are deployed and run in an execution environment, which might be a docker container, operating system, etc. That is specified via *executionEnvironment* property and GIAU uses external repository of the artefacts. Variable *repositoryTag* is an identification of the artefact within the external repository. Additionally some configuration data are stored as well. The structure of the configuration is on the developer. As we explained above the *BlockchainArtefact* is a representation of blockchain node (*BlockckhainNode* class) and the blockchain node is capable of executing a set of blockchain operations (*BlockchainOperation* class). Those classes illustrate how we represent knowledge of blockchain for both discussed levels.

Listing 2. Example of information capturing `Hyperledger-Fabric peer node` as blockchain software artefact

```
{
  name: 'hyperledger-fabric peer',
  implementation: 'hyperledger',
  feature: 'creator',
  executionEnvironment: 'docker',
  imageTag: 'hyperledger/fabric-peer',
  configuration: {
    organization {
      peer_name: "peer1",
      domain: "org1.example.com"
    },
    environment_variables: {
      CORE_PEER_ID: "peer1",
      CORE_PEER_TLS_ENABLED: false,
      CORE_PEER_GOSSIP_USELEADERELECTION: true
          ,
      CORE_PEER_GOSSIP_ORGLEADER: false,
```

```
      ...
    }
  }
}
```

Listing 3. Examples of information capturing blockchain operations carried out by blockchain node

```
{
  bcArtefact: {
    name: "hyperledger-fabric peer",
    ...
  },
  bcOperations: [
    {
      name: "creating a transaction",
      libraryMeta: {
        language: 'nodejs'
        source: "https://fabric-sdk-node.
            github.io/release-1.4/index.html"
      }
    },
    {
      name: "signing a transaction",
      libraryMeta: {
        language: 'nodejs'
        source: "https://fabric-sdk-node.
            github.io/release-1.4/index.html"
      }
    },
    ...
  ]
}
```

3) Capturing infrastructure information: Many papers have been introduced for capturing service information in the edge and cloud. Bellini [17] et al. presented models of infrastructure as a service information in the cloud. Combined with our previous work on IoT Cloud Systems [18], we reuse their model of data-center when modelling edge data-center, model of virtual machines to depict hosts of the blockchain-based applications and model of networks to model connections among nodes of the ES topology. Furthermore, we also consider reuse models of resources in [19], [20] to be able to model IoT devices.

4) Modeling experiments information: When a developer wants to obtain recommendation about the deployments then s/he needs to know topologies, to which blockchain artefacts have been deployed. Knowing the topology is important for the developer because it helps understand various circumstances in which benchmarks have been measured for the topology. Those circumstances include role of blockchain (i.e. whether blockchain has been used for interactions or datastore), how many blockchain artefacts have been deployed, scale of the topology, etc. A topology in GIAU is represented by the *Topology* class. It's a graph composed of instances of *Node* class, presenting graph's nodes. Instance of the *Node* class is a representation of ES component. The edges of the graph represent interactions among the *Node*'s instances (ES components). *SoftwareArtefacts*, including *BlockchainArtefacts* deployed to the ES components. There is a known infrastructure specified via *container* of the *Node* class. The developers can utilize a deployment language, such as as Topology and Orchestration Specification for Cloud Applications (TOSCA), and AWS CloudFormation, to specify the topology. The specification is stored in *specification* attribute and the description language is specified via *specificationLang*. A topology is associated with an experiment.

5) Incorporating benchmark and monitoring data: The developers can utilize blockchain benchmarking and monitoring tools [12] [13] to benchmark blockchain-based applications in ES. The knowledge in our model is extracted from the benchmarks. Since benchmarks (from the developers) might have very different representations, we need to employ data processing techniques (e.g. utilizing a parser utility) to extract benchmarks and ingest the data into our framework. We retrieve data concerning (i) deployment pattern, (ii) software artefact and (iii) infrastructure information from the benchmarks. Then the (i), (ii) and (iii) are queried in a graph database (Neo4J) (to find (i)) and a document database (MongoDB) (to find (ii), (iii)) to determine whether those are known. If so then the benchmarks data are linked to those. Otherwise the data concerning (i), (ii) and (iii) are stored to the graph database and the document database and then the stored data is linked to the benchmarks data. Finally, we take the benchmarks data linked to the (i), (ii) and (iii) and insert them to the document database. The results of the benchmarks in GIAU are represented via quality metrics (*QualityMetric* class, depicted in Figure 2). There are many metrics of quality for blockchain systems, these have been extensively studied in literature ([21], [12], [13]). For example, transaction latency [21] equals confirmation time minus submit time or transaction throughput [21] is computed as total committed transactions divided by total time in seconds. The value of a metric is stored in the *value* property of *QualityMetric*. Furthermore, there are other metrics, like resource utilization and cost, which can be stored.

C. Search and recommendation

GIAU can be used by developers to search for topologies, to which certain blockchain artefacts have been deployed and to see what benchmarks have been measured in those topologies. The developers can also share topologies of their blockchain-based applications to GIAU. Searching the topologies allows the developers to compare their topologies with the benchmarked topologies stored in GIAU. Based on the benchmark information, the developers can infer a recommendation.

We support the developers to look up information representing both levels of knowledge regarding blockchain software artefacts. Our proposed recommendation solution accepts a deployment pattern (derived from the application's topology) as input and utilizes the knowledge stored in GIAU to find deployment pattern, which is most similar to the one submitted on input. Furthermore, the solution has to take quality metrics, which are most relevant to the developer, into consideration. Based on the preferred quality metrics, GIAU looks up benchmarks associated with the most similar deployment pattern and returns a deployment pattern with

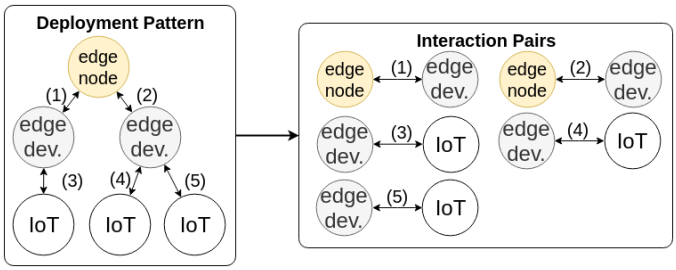

Fig. 3. Obtaining interaction pairs from a deployment pattern

deployed blockchain software artefacts. The returned topology provides a recommendation to the developer about a suitable deployment. Since our goal is to support ES development in the edge and cloud environments, we support the known TOSCA specification of the deployment patterns as input. In order to be able to specify the deployment patterns in TOSCA we derive node types from existing TOSCA Cloudify node types[5]. The derived node types represent different types of nodes of deployment patterns as explained before[6].

Recommendations need to rank a similarity between an input deployment pattern against existing patterns. Nevertheless, the problem of finding the largest common subgraph is known to be NP-hard [22]. That implies that especially for large scale topologies it might take a very long time till a largest common subgraph is obtained. In the current version, we just implement a simple approach. GIAU will iterate over all deployment patterns stored and split each deployment pattern into interaction pairs. An interaction pair is a pair of nodes (representing ES components) connected by an edge in the graph, representing deployment pattern. An example illustrating a transformation of a deployment pattern to a set of interaction pairs is depicted in Figure 3. The deployment pattern for which we find the largest number of matching interaction pairs with the interaction pairs of the submitted deployment pattern is considered as the most similar one. When the most similar deployment pattern is obtained, we search all benchmark experiments associated with the deployment pattern to find the benchmark with best results concerning the preferred quality metrics set by developer.

IV. Prototype and Experiments

A. Prototype

We have implemented a prototype of GIAU with key services shown in Figure 4. The prototype has been developed in Typescript, runs in a NodeJS environment and can be executed inside a docker container. It uses a single MongoDB and a Neo4J database for the repository layer. The implementation and documentation of the prototype is available in the GitHub repository[7].

[5]https://docs.cloudify.co/4.6/developer/blueprints/built-in-types/

[6]Further node types can be found in `giau/tosca_node_types.yaml` of the GitHub repository at https://github.com/rdsea/blockchainbenmarkservice

[7]https://github.com/rdsea/blockchainbenmarkservice/tree/master/giau

B. Experiment data

Real data: We performed evaluations based on our real data from benchmarking Vehicle-to-Everything (V2X) communication scenarios[8]. The real data are stored at `experiments/results/benchmarks_results` of the GitHub repository. We have 324 benchmarks for the topologies with deployed blockchain software artefacts from the real data. There are five entries about different infrastructure's metadata and six related to blockchain software artefacts.

Emulated data To have more data for testing, we generated the emulated data[9]. We generated 250 diverse deployment patterns, of which sizes have been generated randomly by following a normal distribution with $\mu = 100$ and $\sigma = 30$. The types of nodes (refer to Section III) of the deployment patterns are represented by a random variable $X \sim \mathcal{N}(2, 1)$, such that $P(X < 1) = 0.1587$ is the portion of `edge node` node type, $P(1 < X < 2.5) = 0.5328$ of `thing`, $P(2.5 < X < 3.5) = 0.2417$ of `edge dev` and the rest is `cloud`. One of the most complex emulated deployment patterns is composed of 196 nodes (99 `thing` node types, 43 `edge dev`, 41 `edge node`, 12 `cloud`). Topologies, created based on the deployment patterns, are associated with metadata of infrastructures having configurations, which values (CPU core count, storage and memory of containers, latency and bandwidth of network, etc.) are normally distributed. Furthermore, we generated 50 different metadata of blockchain software artefacts. For each deployment pattern we created two different benchmarks results, which values of quality metrics are normally distributed as well. Altogether the emulated data presents more diverse structures and larger scale of the data than we have from the real data.

C. Examples of providing knowledge in blockchain development

Suppose a developer is struggling with selecting a suitable deployment of blockchain software artefacts into the nodes of the deployment pattern. The TOSCA specification for the deployment is given in Listing 4, when developing an ES. The developer utilizes GIAU to obtain recommendations about the deployment. Figure 5 depicts the recommended deployment returned by GIAU[10]. We can observe that the submitted and returned deployment pattern (Figure 5) are similar, but not the same. The returned deployment pattern provides following hints to the developer about the deployment: Deploy `Hyperledger-Fabric peer node` to IoT

[8]The work of benchmarking is out of the scope of this paper. Initial information can be obtained from https://www.researchgate.net/publication/333388734_Benchmarking_Blockchain_Interactions_in_Mobile_Edge_Cloud_Software_Systems.

[9]All emulated data can be found in `giau/tests/data/emulated_data` directory of the above GitHub repository. We implemented a simple utility `giau_emulated_data_generator` published in the GitHub repository

[10]A complete TOSCA .yaml representation of the input and output is given in the directory `giau/tests/data/examples` of the prototype GitHub repository.

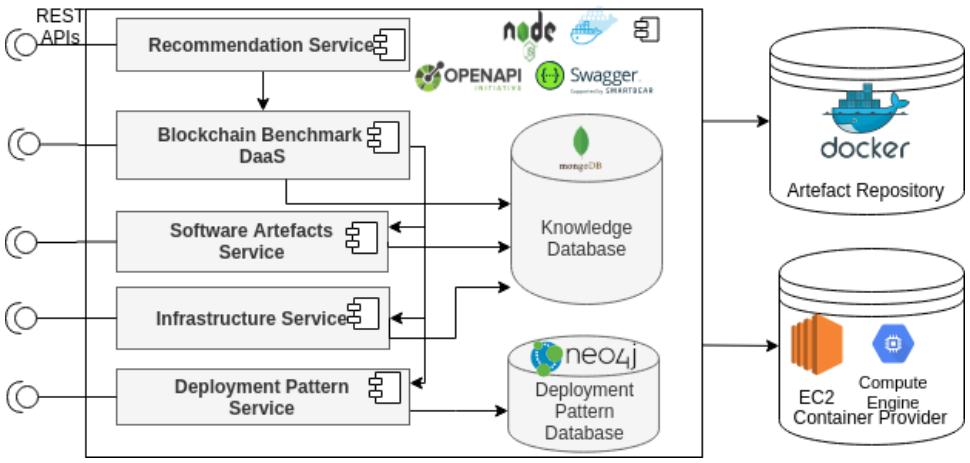

Fig. 4. GIAU Microservice-based prototype

devices, `Hyperledger Fabric orderer node` to edge devices and both `Hyperledger Fabric peer node` and `Hyperledger Fabric order node` with `Apache Kafka` and `Apache Zookeeper` (those are required by Hyperledger Fabric blockchain) to the edge node. Because those Hyperledger Fabric nodes have been deployed to the respective components in the recommended deployment. Beside the deployment, GIAU provides following recommendations (these are not visible in the Figure 5, but are in the TOSCA .yaml representation) regarding hardware configuration of the underlying infrastructure. Use `big machine` for IoT devices and edge node, `small machine`'s configuration for edge devices and a `5G` network configuration for connections among the nodes of the deployment pattern. Exact information regarding the hardware (number of CPU cores, memory, etc.) and network (latency and bandwidth) configurations can be found via the *Infrastructure Service*. We can observe that in this case our recommendation service was able to find a deployment pattern among all in the real data, which is actually most similar to the submitted one. Furthermore, a deployment, for which best benchmarks concerning the preferred quality metrics of the developer, has been returned.

Listing 4. Input example - TOSCA description of a deployment pattern

```
node_templates:
  edge_dev1:
    type: giau.nodes.rsu
    relationships:
      - target: iot1
        type: giau.relationships.nodes_network
  edge_node:
    type: giau.nodes.edge
    relationships:
      - target: edge_dev2
        type: giau.relationships.nodes_network
      - target: edge_dev1
        type: giau.relationships.nodes_network
      - target: iot3
        type: giau.relationships.nodes_network
  edge_dev2:
    type: giau.nodes.rsu
    relationships:
      - target: iot2
        type: giau.relationships.nodes_network
  iot1:
    type: giau.nodes.vehicle
  iot2:
    type: giau.nodes.vehicle
  iot3:
    type: giau.nodes.vehicle
```

D. Performance evaluation

To evaluate performance of GIAU we perform stress testing. We simulate different numbers of concurrent requests being sent to the service and we measure response time (rT) in milliseconds and success rate [23] of the requests by utilizing Apache JMeter[12]. We have performed testing on the following endpoints exposed by the service: (i) sharing knowledge to the service (ii) obtaining recommendation. For both endpoints we have have executed two experiments, first one with the real data and the second one with the emulated data. GIAU has been executed in a docker container on a machine with following hardware configuration Intel Core i7-6820HQ CPU, 16GB RAM memory, Ubuntu 18.04 during the testing.

We performed experiments for stress tests and for recommendation. Shown in Figure 6, `Experiment 1` and `Experiment 2` are for obtaining knowledge from the service by calling a `POST` request to the `/experiment` endpoint of GIAU with real data and with emulated data, respectively, and `Experiment 3` and `Experiment 4` are for recommendation requests by calling requests to `/recommendTopology` with real data and with emulated data, respectively. When we look on those results we can observe that there is a linear dependency between the number of concurrent requests and average rT in `Experiment 1` and `Experiment 2`. But we witness longer rT for the

[11]https://docs.cloudify.co/4.5.0/developer/composer/
[12]https://jmeter.apache.org/

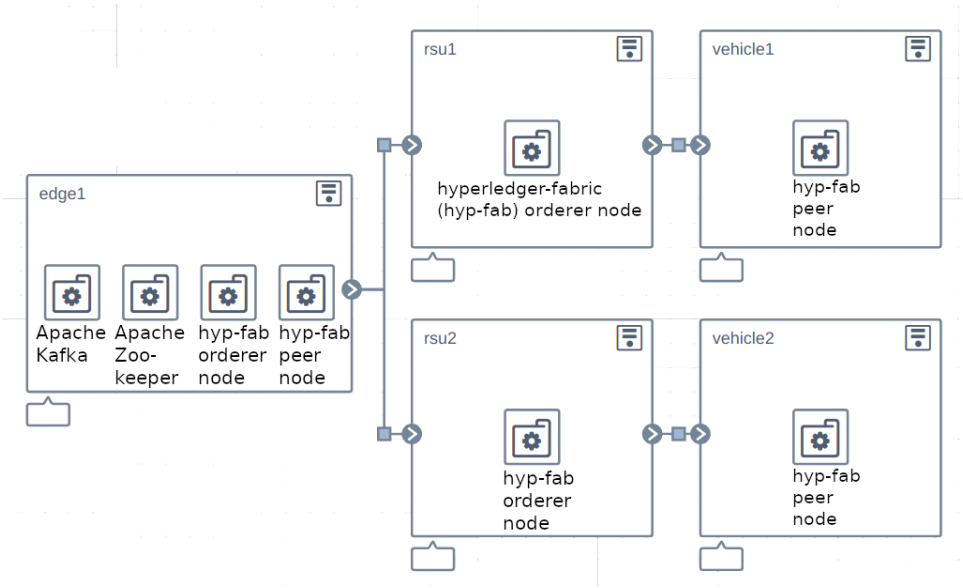

Fig. 5. Deployment information returned by GIAU, depicted via Cloudify Composer[11]

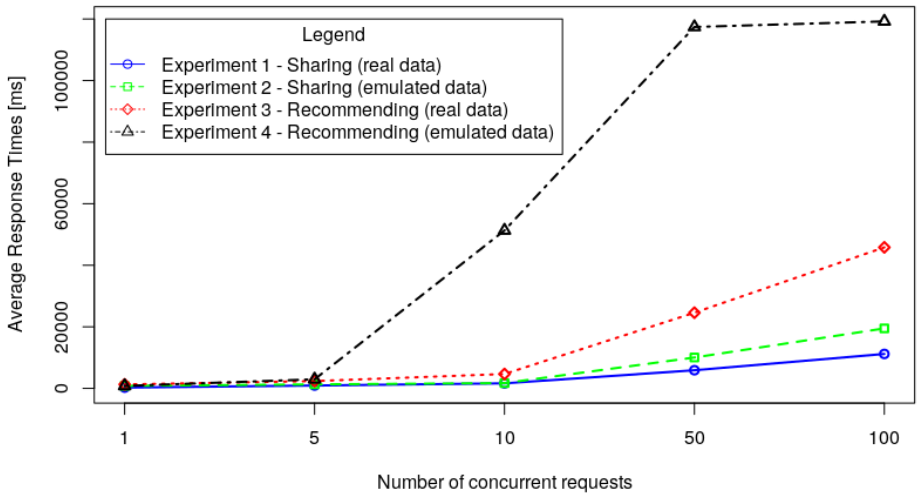

Fig. 6. Results of stress testing - Experiments 1 to 4

`Experiment 2`. That is caused by the diversity and larger scale of the emulated data than we have for the real data.

We observe that in `Experiment 3` GIAU was able to give recommendations even if 100 concurrent requests have been issued and we measured 100% success rate. In `Experiment 4`, we observed that for 50 and 100 concurrent requests GIAU wasn't able to respond on 92% and 98% of requests. In these cases we witness a `HeapOutOfMemory Exception` thrown by the Neo4J database.

E. API for different Tools

External tools with blockchain benchmark and deployment information can use APIs to upload their data into GIAU, thus enriching the knowledge. For example, Figure 7 shows some selected APIs. Currently, we do not provide connectors for extracting information from other tools. However, using various big data ingestion pipelines based on, e.g., Logstash[13], Apache Nifi[14] and Python/JavaScript programs, pushing data into GIAU should be straightforward.

V. Related Work

Knowledge models for IoT and Cloud: The authors of [19] present a unified semantic knowledge base for IoT. The knowledge base models IoT resources (sensors, actuators and physical objects), location information, information about context, policies for dynamic environment and IoT services. However, in [19] they don't provide any knowledge about blockchain in ES as we do in our work. In our work we can reuse their models of IoT resources. Bellini et al. [17] present

[13]https://www.elastic.co/products/logstash
[14]https://nifi.apache.org/

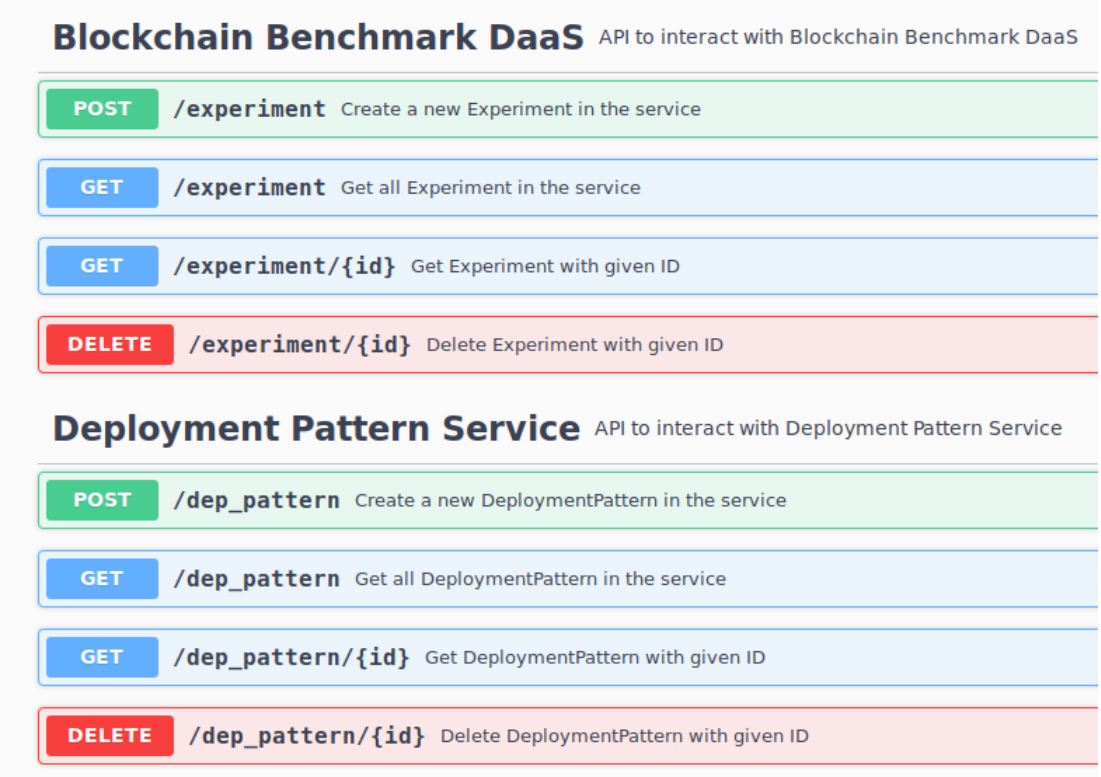

Fig. 7. Example of APIs for integrating with data collection pipelines

a modeling of cloud knowledge by utilizing ontologies to model cloud representations at layers of IaaS, PaaS and SaaS. The authors don't give any blockchain knowledge. However, they provide modelling of IaaS, which we can reuse when specifying infrastructure information in our work.

Design patterns and knowledge models for blockchain Zhang et al. [24] explain potentials of interoperability improvements and challenges in utilizing blockchain to healthcare applications. The authors show that those challenges can be mitigated by applying familiar software patterns as Abstract Factory, Flyweight, Proxy and Publisher-Subscriber. In our work, we focus on blockchain in ES, as opposed to Zhang et al., who concentrate on healthcare applications specifically. In the work by Xu et al. [25] they present design patterns for blockchain-based applications. They utilize the patterns to describe communications of blockchain with external world, to manage data on blockchain, to secure blockchain-based applications, etc. Lu et al. [26] contribute a design pattern as a service platform, which involves data management services and smart contract design. They focus on scalability, privacy and security. The data management services concentrate on scalability and privacy of data in blockchain. The authors of [24], [25] and [26] show application of software patterns on blockchain-based applications but have not provided any blockchain knowledge sharing service as we do in our work.

Blockchain benchmark and monitoring: Several papers have performed benchmarks and testing of blockchain and blockchain networks [12], [27], [28]. Our work complements these blockchain benchmarks and testing as we do not focus on benchmarking but integrate benchmark information.

VI. Conclusions and Future Work

Mastering requirements and performance understanding of blockchain and edge computing systems is challenging for edge service developers. In this paper we have presented a framework for managing performance information about blockchain features and their related deployments for edge services. Knowledge are captured from benchmarks and other information to enable the developer to share and reuse useful information for design and implementation of blockchain-based edge services. Our framework – GIAU – has aggregated various types of data via generic models and provides services for the developer to update, retrieve and search useful blockchain-related performance information.

Currently we work on improving our prototype by employing software engineering techniques to make the performance of the service better. We are also analyzing possibilities to provide a richer evaluation of the prototype. Our future work includes implementation of interfaces to existing benchmarking tools [12], [13] to simplify the gathering of performance information through benchmarks measured by blockchain tools.

Acknowledgment

This work in this paper was partially carried out during Filip Rydzi master thesis at TU Wien, Austria. Results are also

partially reported in his master thesis. We thank Google Cloud Platform Research Credits Program for providing resources for benchmarking and testing.

REFERENCES

[1] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, "Edge-oriented computing paradigms: A survey on architecture design and system management," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 39:1–39:34, Apr. 2018.

[2] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, March 2017.

[3] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3-4, pp. 134 – 155, 2018.

[4] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 15:1–15:13. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134459

[5] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han, "When mobile blockchain meets edge computing: Challenges and applications," *CoRR*, vol. abs/1711.05938, 2017.

[6] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[7] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey," *Sensors*, vol. 18, no. 8, p. 2575, 2018. [Online]. Available: https://doi.org/10.3390/s18082575

[8] M. Samaniego and R. Deters, "Blockchain as a service for iot," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Dec 2016, pp. 433–436.

[9] W. Viriyasitavat, T. Anuphaptrirong, and D. Hoonsopon, "When blockchain meets internet of things: Characteristics, challenges, and business opportunities," *Journal of Industrial Information Integration*, 2019.

[10] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[11] A. Bergmayr, U. Breitenbücher, N. Ferry, A. Rossini, A. Solberg, M. Wimmer, G. Kappel, and F. Leymann, "A systematic review of cloud modeling languages," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 22:1–22:38, Feb. 2018. [Online]. Available: http://doi.acm.org/10.1145/3150227

[12] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1085–1100.

[13] Hyperledger, "Hyperledger caliper: Blockchain performance benchmarking for hyperledger burrow, fabric, iroha and sawtooth," https://hyperledger.github.io/caliper/, [Online; accessed 29-March-2019].

[14] Ethereum, "Ethereum white paper," https://github.com/ethereum/wiki/wiki/White-Paper, [Online; accessed 27-March-2019].

[15] Hyperledger, "Hyperledger architecture, volume 1," https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf, [Online; accessed 23-March-2019].

[16] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 30:1–30:15.

[17] P. Bellini, D. Cenni, P. Nesi, S. Murugesan, and I. Bojanova, "Cloud knowledge modeling and management," in *Encyclopedia of Cloud Computing*. Wiley Online Library, 2016, pp. 640–651.

[18] H.-L. Truong, L. Berardinelli, I. Pavkovic, and G. Copil, "Modeling and provisioning iot cloud systems for testing uncertainties," in *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, ser. MobiQuitous 2017. New York, NY, USA: ACM, 2017, pp. 96–105. [Online]. Available: http://doi.acm.org/10.1145/3144457.3144490

[19] S. N. A. U. Nambi, C. Sarkar, R. V. Prasad, and A. Rahim, "A unified semantic knowledge base for iot," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, March 2014, pp. 575–580.

[20] D. Le, N. C. Narendra, and H. L. Truong, "HINC - harmonizing diverse resource information across iot, network functions, and clouds," in *4th IEEE International Conference on Future Internet of Things and Cloud, FiCloud 2016, Vienna, Austria, August 22-24, 2016*, M. Younas, I. Awan, and W. Seah, Eds. IEEE Computer Society, 2016, pp. 317–324. [Online]. Available: https://doi.org/10.1109/FiCloud.2016.52

[21] Hyperledger Foundation, "Hyperledger blockchain performance metrics," https://www.hyperledger.org/wp-content/uploads/2018/10/HL_Whitepaper_Metrics_PDF_V1.01.pdf, [Online; accessed 26-May-2019].

[22] T. Akutsu, "A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 76, no. 9, pp. 1488–1493, 1993.

[23] L. O'Brien, P. Merson, and L. Bass, "Quality attributes for service-oriented architectures," in *Proceedings of the International Workshop on Systems Development in SOA Environments*, ser. SDSOA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–.

[24] P. Zhang, J. White, D. C. Schmidt, and G. Lenz, "Design of blockchain-based apps using familiar software patterns to address interoperability challenges in healthcare," in *PLoP-24th Conference On Pattern Languages Of Programs*, 2017.

[25] X. Xu, I. Weber, and M. Staples, *Architecture for Blockchain Applications*. Springer International Publishing, 2019, ch. Blockchain Patterns.

[26] Q. Lu, X. Xu, Y. Liu, and W. Zhang, "Design pattern as a service for blockchain applications," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov 2018, pp. 128–135.

[27] M. A. Walker, A. Dubey, A. Laszka, and D. C. Schmidt, "Platibart: A platform for transactive iot blockchain applications with repeatable testing," in *Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things*, ser. M4IoT '17. New York, NY, USA: ACM, 2017, pp. 17–22.

[28] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey, "Measuring ethereum network peers," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: ACM, 2018, pp. 91–104.