A Blockchain-based Decentralized Self-balancing Architecture for the Web of Things

Aleksandar Tošić[1,2][0000−0001−5627−4420], Jernej Vičič[2][0000−0002−7876−5009], and Michael Mrissa[1,2][0000−0002−2330−1004]

[1] InnoRenew CoE,
Livade 6, 6310 Izola, Slovenia
{firstname.surname}@innorenew.eu
[2] University of Primorska,
Faculty of Mathematics, Natural Sciences and Information Technology,
Glagoljaška ulica 8, 6000 Koper, Slovenia
{firstname.surname}@famnit.upr.si

Abstract. Edge computing is a distributed computing paradigm that relies on the computational resources of end devices in a network to bring benefits such as low bandwidth utilization, responsiveness, scalability and privacy preservation. Applications range from large scale sensor networks to IoT, and concern multiple domains (agriculture, supply chain, medicine, etc.). However, resource usage optimization is a challenge due to the limited capacity of edge devices and is typically handled in a centralized way, which remains an important limitation. In this paper, we propose a decentralized approach that relies on a combination of blockchain and a consensus algorithm to monitor network resources and, if necessary, migrate applications at run-time. We integrate our solution into an application container platform, thus providing an edge architecture capable of general purpose computation. We validate and evaluate our solution with a proof-of-concept implementation in a national cultural heritage building.

Keywords: Edge computing · Internet of Things · Decentralized applications · Blockchain

1 Introduction

In the last few years, edge computing has received a lot of attention as an alternative to cloud computing, due to the multiple advantages it offers, such as low bandwidth usage, responsiveness, scalability [10], and privacy preservation [17]. Edge computing has becomes possible due to the evolution of devices that offer more computational power than ever. Combined with application container platforms such as Docker [3] that mask heterogeneity problems, it becomes possible for connected devices to form a homogeneous distributed run-time environment. Additionally, orchestration engines (i.e., Kubernetes[3]) have been developed that

[3] https://kubernetes.io/

manage and optimize usage of network, memory, storage, or processing power for edge devices and improve the global efficiency, scalability and energy management of edge platforms. However, such solutions are centralized, which means that they represent a single point of failure (SPOF), which entails several drawbacks, such as lack of reliability and security. The problem is so critical that developments for high availability have been explored, for instance with Kubernetes[4].

This paper proposes a solution that uses a decentralized algorithm that monitors network resources to drive application execution to address this problem. Our solution relies on an original combination of blockchain, a consensus algorithm, and a containerized monitoring application to enable run-time migration of applications, when relevant, according to the network state. It provides several advantages, such as verifiable optimal usage of all devices on the network, better resilience to disconnection, independence from cloud connection, improved privacy and security.

The remainder of this paper is organized in 7 sections. Section 2 introduces our motivating scenario related to a cultural heritage building and shows the need for a decentralized approach. Section 3 overviews relevant related work and highlights the originality of our approach. Section 4 details our proposed architecture and shows how it drives run-time migration of applications on the edge. Section 4.2 presents our network monitoring application and shows how monitoring takes place. In Section 5, we propose a technical implementation, and we validate and evaluate our solution with a proof-of-concept prototype related to our cultural heritage scenario. Section 6 discusses the results obtained and gives insights for possible future work.

2 Motivating Scenario

In this section, we illustrate the relevance of our approach with a scenario related to a Slovenian cultural heritage building located in Bled, Slovenia. This building has been equipped with multiple sensors to monitor its dynamic environment that affects the building and its contents. The collected data includes temperature, CO2, relative humidity, Volatile Organic Compounds (VOC), ambient light and atmospheric pressure. In this scenario, the following constraints motivate the need for a fully decentralized edge computing approach:

- Privacy: collected data about the state of the technological solution being deployed is classified as sensitive information. Although data about the building could be sent to the cloud, data about the state of resources needs to remain local and only accessible for administration purpose and for the deployed solution to self-manage.
- Reliability: centralized orchestration is not appropriate as data collection needs to be resilient to failure of any device. The network of devices needs to adjust to device disconnection at any time and keep operating in an optimal way.

[4] `https://kubernetes.io/docs/setup/independent/setup-ha-etcd-with-kubeadm`

- Cost: reducing the overall cost by avoiding investing in a cloud infrastructure that involves monthly payments and permanent connection to maintain.
- Scalability: as the number of devices will evolve over time, it is necessary for the solution to be able to adjust to changes and homogeneously spread the computation over the network.
- Performance: reactivity to external events is improved if processing is performed on-site.
- Cost effectiveness: using existing devices that control sensors to perform necessary processing reduces the resource requirements of cloud based solutions, which reduces cost.

In this context, it is relevant to equip devices with the capacity to run applications locally and to self-manage the global network load and distribute it over connected devices, according to the state of the network. In the next section, we present related work and show the need for a decentralized self-managed platform on the edge. We also overview existing solutions to abstract from platform heterogeneity and justify the technological choice of a container platform to support our solution.

3 Background Knowledge and Related Work

A recent study by Taherizadeh et al. [19] shows that no widely-used cloud monitoring tools yet provide an integrated monitoring solution within edge computing frameworks, as some monitoring requirements have not been thoroughly met by any of them. Diallo et al. [6] present AutoMigrate, which incorporates a selection algorithm for deciding what services to migrate that maximizes the availability of migration. The system addresses most of the problems that are discussed in our paper. However, it relies on a single agent to manage services introducing a Single Point Of Failure (SPOF). The most notable difference in our implementation is a decentralized architecture that eliminates the SPOF.

3.1 Choreography solutions for edge computing

Strictly observing the definition of orchestration, it always represents control from one party's perspective. This differs from choreography, which is more collaborative and allows each involved party to describe its part in the interaction [16]. However, to the authors' knowledge, there are no choreography solutions that tackle the problems defined in the previous section. Existing orchestration solutions typically rely on a master/slave model where a node is put in charge of the network and decides to allocate applications to nodes according to an optimization algorithm.

Containers as used in the purpose of this paper are run as a group of namespaced processes within an operating system, avoiding the overhead of starting and maintaining virtual machines (at the same time providing most of the functionalities).

The selected platform for our research was Docker [3] as it is the most widely used platform and one of the few that can migrate apps at runtime and enables easy communication. The migration is done by pausing the container, dumping the context of the paused container, transferring the context on a different host that can resume the execution given the context.

3.2 Decentralized Self-managing IoT Architectures

Kubernetes [8] is the most widely used orchestration tool, it is the go-to tool for orchestration in the Google cloud, and is the most used in the Microsoft Azure platform and similar products. It is also the most feature-filled orchestration tool available [12]. It has strong community support across many different cloud platforms (in addition to Google cloud, OpenStack, AWS, Azure).

AWS Elastic Container Service (AWS ECS) [1], Amazon's native container orchestration tool, is the best option for orchestration of AWS services as it is fully integrated into the Amazon ecosystem. It thus integrates easily with other AWS tools. The biggest limitation is that it is limited to Amazon services.

Docker Swarm [5] ships directly with Docker (integrates with Docker-compose) and is supposed to have the simplest configuration. However, it lacks some advanced monitoring options as compared to other products like Kubernetes.

Apache Mesos' based DC/OS [6] is a "distributed operation system" running on private and public cloud infrastructure that abstracts the resources of a cluster of machines and provides common services.

All presented architectures still have a common flaw: single point of failure and a lack of integration with edge computing.

There have been some proposed solutions that enable fully decentralized self-managing architectures for the IoT. For example, [11] focuses on a decentralized solution for energy management in IoT architectures connected to smart power grids. In [7], the authors propose a distributed IoT approach for electrical power demand management problems based on "distributed intelligence" rather than "traditional centralized control," with the system improving on many levels. Souzdalenko et al. [18] further develop the former approach by creating a decentralized distributed model of an IoT; where consumers can freely join and leave the system automatically at any time. Niyato et al. [13] present a system that uses machine-to-machine (M2M) communication to reduce the cost of a home energy management system. A distributed and decentralized microscopic simulation that eliminates the central entity and thus avoids the bottleneck in synchronization is presented in dSUMO [4]. In [2], the authors demonstrate the effectiveness of utilizing a publish/subscribe messaging model as connection means for indoor localization utilizing Wireless Sensor Networks (WSNs) through a middle-ware, the results showed that RSS reaches an acceptable level of accuracy for multiple types of applications.

[5] `https://github.com/docker/swarm`
[6] `https://dcos.io/`

However, all the aforementioned contributions are different from the solution we propose in this paper, at two levels. First, they mostly focus on a single specific aspect and find an optimal solution for it, without considering the fact that an IoT architecture involves multiple criteria that require optimization. In our work, we already consider multiple criteria to optimize application migration, while envisioning that this number of criteria can increase in the future. Second, as far as we know, there is no approach that combines a blockchain data structure with a consensus algorithm in a single framework with the objective to drive application migration at run-time on the edge, which is the main contribution of this paper.

4 A Decentralized Self-managing Architecture

In the following, we describe the general architecture that support our edge computing platform. Devices on the edge are nodes running node software and containerization software. A node can join the network by following a network protocol for exchanging known nodes and participate by executing the consensus algorithm. Nodes keep discovering the network by asking connected nodes for peers. For the sake of simplicity, in this paper we consider that the number of nodes remains reasonably limited, so that large scale discovery issues remain out of the scope of this paper.

4.1 General Architecture

Our devices are equipped to allow a specific containerized application (called node app) to introspect the state of the node and handle the diffusion of this information over the network. It also is responsible for maintaining the information about the other nodes up to date, for participating in the consensus algorithm, and for listening to messages coming from the exposed node API.

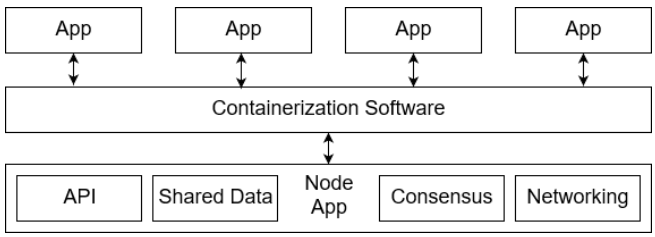


Figure. 1: Architecture of an edge device software platform: a Node App that deals with the consensus algorithm, accesses shared data and exposes the querying API is deployed into the container (in our case Docker).

Figure 1 shows the key components of Nodes in the system. The node software is deployed into the container, in our case Docker. The container mounts a direct

socket to the containerization service for querying the state of the system and managing local containers. Docker is useful here to alleviate from the typical heterogeneity problems encountered in the IoT world (different processors and OSes).

4.2 Node Application

Every 500 milliseconds, each device collects information about the state of its neighbours. Typically, a state is a vector of scores that describes the device state and the applications being executed by the node. In this work we define a state to be a matrix of vectors

$$S(APP, CPU, RAM, DISK, NETWORK, TIMESTAMP)$$

where each vector represents an application being executed by the node and the corresponding resource consumption. Resources are reported as a fraction of the total available. In order to have comparable values between nodes, reporting on CPU usage and network utilization the CPU is normalized with the number of cores whereas network bandwidth(download/upload) is measured when transferring containers between nodes.

Monitoring resources within the P2P network is done by having nodes maintain a list of scores of all other nodes (neighbours or not). All nodes periodically send digitally signed messages containing their score to all neighbour nodes. All nodes follow simple P2P broadcasting rules that guarantee finality and efficiency in message propagation.

- If elapsed time greater then ΔST, send signed a message containing own score to all neighbour nodes.
- When receiving a new score message, check if the message was received before (compare digital signatures).
- If the message was not seen before, send it to all connected nodes with the exception of the originating node.

Where ΔST is the time interval in which the container statistics are collected and it is configurable and should depend on the time interval of the consensus algorithm. The score pool hence contains scores of all nodes participating in the network. Each score has a corresponding time-stamp which is later used by elected nodes to create a migration strategy.

Messages containing blocks can become relatively large when the number of applications in the system increases. For improved efficiency, every score message broadcast is prefaced with a "Do you need this" (DYNT) message coupled with the digital signature of the message only. Messages are sent to nodes that reply to the DYNT message to minimize bandwidth use.

Consensus algorithm The network requires a consensus algorithm to avoid race conditions when migrating applications. The choice of a consensus algorithm

depends on the requirements of the implementation and domain of application. In general, any consensus based on leader election can be plugged in. Examples of such consensus algorithms are Paxos [9], Raft [15], PoET [14], etc. However, in our implementation PBFT [5] was used as it is relatively simple to implement and all its properties satisfy our demands. The only real drawback of the algorithm is that the number of messages increases exponentially with the number of nodes, so it is not applicable to large networks. It was a viable alternative for our proof-of-case implementation with a limited number of nodes. The elected leader is responsible for creating a migration plan and including the resource consumption estimates in a block. The block gets digitally signed so other nodes can verify it originates from the elected leader. Nodes receiving a new block must verify the migration plan by computing it locally and comparing the results. If the migration plan is equal, they act on it, otherwise discard the block and wait for a new one. With these simple protocol rules in place the network is Byzantine fault tolerant [5]. The block verification step is necessary to minimize accidental network forks. A migration strategy is analogous to blocks in block-chain based systems. Blocks contain all the data shared among nodes in the network and include a digital signature of the previous block thus creating a block chain. In order to create a digital signature of block $n+1$ a node needs to have the digital signature of node n. A well formed block can be verified by other nodes that also have block n. In case of a malformed block, verification will fail, and nodes will reject the block, thus forcing the nodes to agree on the shared data. The block serves as an instruction set mapping applications to nodes. Consider a case with 4 nodes in set N denoted by A, B, C, and D respectively. All nodes share their score and keep a local copy of reported scores of other nodes. Each node stores a vector of applications $v \in V$ that need to be executed. Each node has a canonical list of block B of size k where k is the current block height. Table 1 shows an example of a block k which assigns every $v \in V$ to a node $n \in N$ To create block $k+1$ a node elected as leader computes an assignment such that the use of resources is optimized (improved). The input to the Algorithm 1 is limited to block data to ensure determinism that can enforce consensus. The Algorithm 1 depends on the application domain and exploring available possibilities will be subject to future work. In this paper, we use the simple described in Algorithm 1, which is deterministic and can only take the block data as input for computation. Once a block is created, currently reported scores are included that will be used to compute block $k+2$. Additionally, blocks are equipped with meta-data like block hash, previous block hash, etc. to facilitate their utilization.

5 Implementation and Evaluation

5.1 Technical Implementation

As described in Section 2, we have implemented and evaluated our solution with a set of sensors deployed in the cultural heritage building Mrakova Domačija in

Data: BlockData
Result: Migration plan
$Max \leftarrow FindMaxLoadedNode(BlockData)$;
$Min \leftarrow FindMinLoadedNode(BlockData)$;
if *!AppQueue.isEmpty()* **then**
  while *!AppQueue.isEmpty()* **do**
    $Min \leftarrow FindMinLoadedNode(BlockData)$;
    $Min.addApp(AppQueue.dequeue())$;
  end
else
  $AppToMigrate \leftarrow Max.MaxLoadApp$;
  $CurrentDeltaScore \leftarrow (Max.score - Min.score)$;
  $FutureDeltaScore \leftarrow$
   $(Max.score - AppToMigrate.score) - (Min.score + AppToMigrate.score)$;
  if $Math.abs(CurrentDeltaScore > FutureDeltaScore)$ **then**
    Migrate $AppToMigrate$ to Min;
  end
end

Algorithm 1: Deterministic migration plan generation algorithm

Table 1: Block data

V	Node	RAM	DISK	CPU	Average Latency
v_0	A	50%	23%	90%	23ms
v_1	B	47%	87%	23%	33ms
v_2	C	12%	25%	15%	51ms
v_3	A	35%	14%	56%	101ms
v_4	D	25%	74%	16%	9ms

Bled, Slovenia. Each sensor is connected to a Raspberry Pi device that hosts a Linux Alpine OS in a Docker container. The container has access to the docker daemon via unix socket. We developed our node application inside a container, it relies on the Docker introspection capacity (`docker stats` command called from our Java program) to collect information about each device. The devices simply collect temperature and relative humidity measurements and calculate their averages. It also hosts a HTTP server[7] that exposes a RESTful API providing access to the system. In such a decentralized system, interaction can be done by any node in the network as follows:

– HTTP GET gives a representation of the target node, which includes information about the state of the device as well as all the necessary information about the node (i.e., last connection time, average connection time, etc.). HTTP GET enables users to view the shared pool of resource stats nodes maintain. Most importantly, it gives a list of all applications in the system.

[7] Please note that CoAP could be used for energy saving purposes.

– HTTP PUT/POST enables users to queue an application to be run by the system.
– HTTP DELETE is utilized when an application must be deleted from the queue.

In order to deploy our prototype, we use 5 Raspberry Pi 3 Model B+ connected to Arduino Nano via USB (Universal Serial Bus), the Arduino is connected to the sensors via UART (Universal Asynchronous Receiver Transmitter) ports. We have connected DHT22 sensors to the Arduino boards to capture temperature and humidity.

5.2 Validation and Evaluation

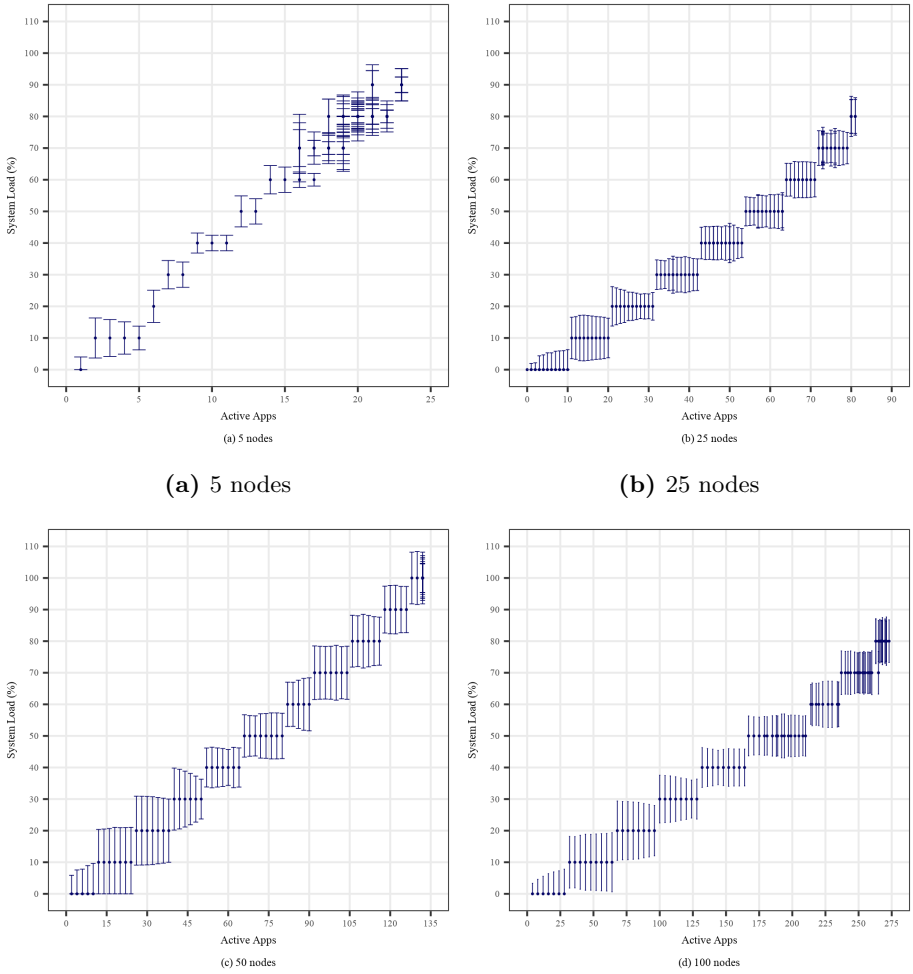
To validate the feasibility of our approach and test its scalability we ran performance simulation test cases. In each test case, a fixed number of nodes formed a P2P network. Nodes were assigned applications to execute. Each application had a random execution time and preset resource consumption expressed in fractions between 5% - 40%. For the sake of simplicity, only one resource was used (CPU). The simulation ran for 100 blocks with a block time of 1 second. Applications were queued until the average load of the entire system rose above 90%. The migration strategy was implemented based on the algorithm described in Section 4.2. Applications arrived in the queue with certain probability, which was gradually increased with the number of nodes in the system. From the reported resource loads of nodes (reported in %), we compute the standard deviation as a measure of how balanced resource consumption is.

In Fig. 2, we observe that the standard deviation remains low even when the number of applications in the system grows. The lower load cases where we can observe higher swings in standard deviations are expected due to the low number of applications in the system. The crossover happens when the number of applications exceeds the number of nodes and migrations can be beneficial. Below the threshold, there are bound to be nodes that do not run any applications. We can observe from Fig. 2a that as the number of nodes is low, resource balancing between nodes is effective earlier, which explains why the measures are less marked than with the other figures, that correspond to test cases where it takes the simulation a longer time to reach the point of crossover where a higher number of applications is distributed over a lower number of nodes.

Figures 2b, 2c, 2d show that the architecture can scale with the growing number of nodes in the network. Additionally, the naive algorithm for creating a migration strategy performed well in distributing load across the system.

6 Discussion and Conclusion

In this paper, we propose a decentralized solution to the resource usage optimization problem, a typical issue in edge computing. Our solution avoids the single



(a) 5 nodes **(b)** 25 nodes

(c) 50 nodes **(d)** 100 nodes

Figure. 2: Simulation results, error bars are standard deviation of the system load

point of failure that centralized architectures suffer from and improves network resilience as it does not depend on a master node. To design our solution, we have combined a blockchain shared data structure and a consensus algorithm with a monitoring application that runs on top of the Docker platform. Such combination allows edge devices to check at run-time if there is a need for migrating an application, and to reach consensus on a decision to do so. With our contribution, edge devices become a completely decentralized and distributed

run-time platform. We have implemented and evaluated our solution with a set of sensors deployed in a cultural heritage building in Bled, Slovenia.

Results show that our approach is able to adjust and normalize the application load over a set of nodes. It also provides, thanks to the fact that the algorithm we use is deterministic and that all the data is stored in a distributed structure, the possibility to verify all the decisions that have been taken to optimize the usage of edge devices. The consensus algorithm that we use also allows adjustments to the global network behaviour for entering or leaving nodes.

Several limitations have been identified that give insights for future work. First, it is important to observe how adding and removing devices affects network behaviour and to explore how scalable our approach is over a large number of devices. Second, it seems appropriate to find out what specific aspects of use cases can help determine which consensus algorithm is most suitable for deploying our solution, in order to best match the use case requirements. Third, it includes semantically describing applications and the services that edge devices offer, to support application migration, and combine in the same architecture the need for efficiently managing network resources together with the needs of applications in terms of functionality and quality of service.

7 Acknowledgment

The authors gratefully acknowledge the European Commission for funding the InnoRenew CoE project (Grant Agreement #739574) under the Horizon2020 Widespread-Teaming program and and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Union of the European regional Development Fund). The first author also acknowledges the support of the ARRS grant N1-0093.

References

1. Acuña, P.: Amazon ec2 container service. In: Deploying Rails with Docker, Kubernetes and ECS, pp. 69–98. Springer (2016)
2. Al-Madani, B.M., Shahra, E.Q.: An energy aware plateform for iot indoor tracking based on rtps. Procedia computer science **130**(C), 188–195 (2018)
3. Anderson, C.: Docker [software engineering]. IEEE Software **32**(3), 102–c3 (2015)
4. Bragard, Q., Ventresque, A., Murphy, L.: Self-balancing decentralized distributed platform for urban traffic simulation. IEEE Transactions on Intelligent Transportation Systems **18**(5), 1190–1197 (2017)
5. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186 (1999)
6. Diallo, M.H., August, M., Hallman, R., Kline, M., Slayback, S.M., Graves, C.: Automigrate: a framework for developing intelligent, self-managing cloud services with maximum availability. Cluster Computing **20**(3), 1995–2012 (2017)
7. Higgins, N., Vyatkin, V., Nair, N.K.C., Schwarz, K.: Distributed power system automation with iec 61850, iec 61499, and intelligent control. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **41**(1), 81–92 (2011)

8. Hightower, K., Burns, B., Beda, J.: Kubernetes: Up and Running: Dive Into the Future of Infrastructure. " O'Reilly Media, Inc." (2017)
9. Lamport, L., et al.: Paxos made simple. ACM Sigact News **32**(4), 18–25 (2001)
10. Mach, P., Becvar, Z.: Mobile edge computing: A survey on architecture and computation offloading. IEEE Communications Surveys & Tutorials **19**(3), 1628–1656 (2017)
11. Maior, H.A., Rao, S.: A self-governing, decentralized, extensible internet of things to share electrical power efficiently. In: 2014 IEEE International Conference on Automation Science and Engineering (CASE). pp. 37–43. IEEE (2014)
12. Medel, V., Rana, O., Bañares, J.Á., Arronategui, U.: Modelling performance & resource management in kubernetes. In: 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC). pp. 257–262. IEEE (2016)
13. Niyato, D., Xiao, L., Wang, P.: Machine-to-machine communications for home energy management system in smart grid. IEEE Communications Magazine **49**(4), 53–59 (April 2011). https://doi.org/10.1109/MCOM.2011.5741146
14. Olson, K., Bowman, M., Mitchell, J., Amundson, S., Middleton, D., Montgomery, C.: Sawtooth: An introduction. The Linux Foundation, Jan (2018)
15. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). pp. 305–319 (2014)
16. Peltz, C.: Web services orchestration and choreography. Computer **36**(10), 46–52 (Oct 2003)
17. Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017)
18. Suzdalenko, A., Galkin, I.: Instantaneous, short-term and predictive long-term power balancing techniques in intelligent distribution grids. In: Doctoral Conference on Computing, Electrical and Industrial Systems. pp. 343–350. Springer (2013)
19. Taherizadeh, S., Jones, A.C., Taylor, I., Zhao, Z., Stankovski, V.: Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. Journal of Systems and Software **136**, 19–38 (2018)