SyncMesh: Improving Data Locality for Function-as-a-Service in Meshed Edge Networks

Daniel Habenicht
Technical University Berlin
Berlin, Germany
daniel.habenicht@campus.tu-berlin.de

Kevin Kreutz
Technical University Berlin
Berlin, Germany
kreutz@campus.tu-berlin.de

Soeren Becker
Technical University Berlin
Berlin, Germany
soeren.becker@tu-berlin.de

Jonathan Bader
Technical University Berlin
Berlin, Germany
jonathan.bader@tu-berlin.de

Lauritz Thamsen
University of Glasgow
Glasgow, United Kingdom
lauritz.thamsen@glasgow.ac.uk

Odej Kao
Technical University Berlin
Berlin, Germany
odej.kao@tu-berlin.de

ABSTRACT

The increasing use of Internet of Things devices coincides with more communication and data movement in networks, which can exceed existing network capabilities. These devices often process sensor or user information, where data privacy and latency are a major concern. Therefore, traditional approaches like cloud computing do not fit well, yet new architectures such as edge computing address this gap. In addition, the Function-as-a-Service (FaaS) paradigm gains in prevalence as a workload execution platform, however the decoupling of storage results in further challenges for highly distributed edge environments.

To address this, we propose SyncMesh, a system to manage, query, and transform data in a scalable and stateless manner by leveraging the capabilities of Function-as-a-Service and at the same time enabling data locality. Furthermore, we provide a prototypical implementation and evaluate it against established centralized and decentralized systems in regard to traffic usage and request times.

The preliminary results indicate that SyncMesh is able to exonerate the network layer and accelerate the transmission of data to clients, while simultaneously improving local data processing.

CCS CONCEPTS

• **Computer systems organization → Peer-to-peer architectures**; **Cloud computing**.

KEYWORDS

mesh network, edge computing, fog computing, data locality, data management, function-as-a-service

ACM Reference Format:
Daniel Habenicht, Kevin Kreutz, Soeren Becker, Jonathan Bader, Lauritz Thamsen, and Odej Kao. 2022. SyncMesh: Improving Data Locality for Function-as-a-Service in Meshed Edge Networks. In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys'22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526275

EdgeSys'22, April 5–8, 2022, RENNES, France

1 INTRODUCTION

With the increase in computational power and popularity of smart mobile devices, the Internet of Things (IoT) has been gaining momentum in the past years. In turn, aspects like scalability [12] and locality of data [25] are becoming more challenging [7]. Therefore, cloud infrastructures are getting increasingly distributed to support localized edge and fog architectures that manage requests in close proximity to the users and data [23]. These architectures reduce response latencies and at the same time unburden the network layer [17, 20]. In order to cope with the increasing scale and the dynamic nature of IoT environments, ongoing research [4, 14] proposes to integrate ad-hoc mesh networks and decentralized capabilities to further enhance the scalability and resilience of edge and IoT architectures. In addition, the Serverless and FaaS paradigm has been identified as a suitable computing model for the IoT [18], since it simplifies the deployment at the edge, enables on-demand allocation of resources, and allows for the execution of tasks in lightweight containers [19].

In the FaaS paradigm, computation and storage are typically decoupled: The functions themselves are stateless and the used data or results are stored in distributed storage systems. Although distributed file systems (DFS) or object stores like S3 simplify the usage of FaaS in cloud computing environments, the dynamic nature of upcoming architectures in the edge-cloud continuum poses new challenges: Devices are often connected via an unreliable and possibly slow network connection, only provided with limited hardware resources, and can join and leave the network at any time in an ad-hoc manner [11].

Summarizing, Serverless and FaaS enable a scalable workload execution, but rely on external storage to store and retrieve state, typically outsourced to centralized cloud systems [10]. The possible impact of the external storage for FaaS and Serverless has been identified as a major issue in cloud architectures [13], yet is even further exacerbated in dynamic IoT and edge environments.

In an effort to address this challenge, this paper presents *SyncMesh*, a new system that improves data locality in mesh-based edge and fog computing environments for FaaS workloads. Instead of moving all generated data of edge devices, i.e. sensor readings, to a central cloud storage, the information is stored on the respective nodes, analyzed, aggregated and transformed locally, and provided to other nodes in the network on-demand. Our prototype implementation of SyncMesh offers a FaaS interface for local data processing and is able to exchange data between participating nodes in a sensor network, significantly reducing network transmissions in the highly distributed and dynamic edge/fog environments.

In summary, the main contributions of this paper are:

- We outline assumptions, challenges and requirements of data storage and locality in highly distributed fog and edge environments.
- We derive the *SyncMesh* architecture and implement a prototype[1] to tackle the identified challenges and enable data locality as well as in-situ data processing.
- We present a preliminary evaluation of our prototype by implementing a relevant use case using real world data and comparing our results to different baseline scenarios.

The remainder of this paper is structured as followed: We first provide an overview of the related work before stating our assumptions and discussing the requirements. Subsequently, we present the *SyncMesh* prototype, conducted experiments as well as results before concluding the paper.

2 RELATED WORK

In general, the importance of storage and data management for the upcoming distributed environments is growing, not only due to the ever increasing generated data [16]. Subsequently, several approaches to improve the storage layer in fog and edge computing environments exists: Many systems focus on efficient cloud offloading [3, 26], in which advanced algorithms are used to time the offloading of data to the cloud efficiently [22]. In contrast to this, with *SyncMesh* our goal is to store and analyze the data where it is created.

In [9], the authors evaluate the performance of different object store system in edge and fog environments, while also identifying storage as a major challenge. In subsequent works, Confais et. al.[8] propose a combination of a network attached storage and peer-to-peer technology to improve the performance of object stores in the aforementioned environments. Although presenting promising results, their approach still relies on a distributed storage layer available to the respective edge nodes, whereas *SyncMesh* aims to improve the data locality.

More similar to our approach, other works avoid the use of central instances entirely and rely on P2P communication between the devices [24]. Mayer et al. [15] address data management problems within decentralized fog networks and introduce their own solution called FogStore, which is a context-aware distributed data storage system. In addition to state management, the paper outlines specific replica placement strategies and a generalized API for querying and manipulating data. In contrast, in our approach we neglect replicas and rather only exchange data between nodes if needed.

Furthermore, the Nebulastream platform [27] was designed with the specific purpose of usage as a data management system for the IoT and addresses issues arising from centralized approaches. They rely on stream processing instead of FaaS.

In regard to FaaS, the Fog function project [5] explores FaaS as a serverless programming model within the context of IoT and uses event-listeners to discover and orchestrate devices and resources in IoT environments. In addition, Fog function was integrated into FogFlow, which is a framework for fog computing with the goal of providing a programming model that enables programmers to develop IoT systems more easily [6].

Although the FaaS paradigm is often highlighted as fitting workload and data analytics platform for fog and edge computing [18], most are focused on i.e. enabling a lightweight orchestration [19] or scheduling based on resource availabilities [21] and often not take the location of data into account. With *SyncMesh* we attempt to address this challenge, to further improve the data locality.

3 SYNCMESH

In this section, we present the design of the *SyncMesh* system. Firstly, we describe assumptions about the computing environment before we derive a set of requirements for our system. Subsequently, the different components of the *SyncMesh* system are introduced.

3.1 Assumptions

In order to define the scope and envisioned use case of *SyncMesh*, we make the following assumptions about the expected environment: We assume a highly distributed environment in the edge-cloud continuum, consisting of lightweight and heterogeneous devices. The devices are interconnected via peer-to-peer connections in an ad-hoc mesh network and act as autonomous nodes. Therefore, the environment is not administered or managed by a central entity, and devices organize themselves in, i.e. swarms, in order to enable scenarios in the context of remote sensing and environmental awareness [11]. In the related work, such architectures are often proposed to increase the scalability and reliability of fog and edge computing environments and to cope with the dynamic nature of the IoT.

Furthermore, we assume that the devices are equipped with sensors such as temperature and air quality sensors or audio and video sources, resulting in data directly created on the respective nodes. The devices are not constrained to data center boundaries and can be located anywhere in i.e. a smart city or a wildlife refuge [1], and due to the possibly unreliable and slow network it is not feasible to constantly synchronize data between nodes or upload all sensor data to the cloud.

Finally, for our initial prototype we presume that inside the network of edge devices, a discovery process for neighboring nodes in the mesh exists. Therefore, for the remainder of this paper we expect that devices can join the network and automatically discover other nodes in the proximity, but neglect an actual implementation and refer to the related work. For instance, previous work [2] has utilized metrics provided by the underlying mesh network to identify direct neighbors in the ad-hoc swarm.

3.2 Requirements

From our assumptions, we derive the following requirements for our system:

Support operation during unreliable network conditions. The described edge sites are often distributed across several remote locations and are only connected with limited bandwidth and high latencies. In addition, network partitions and outages are more common than in traditional data centers. Therefore, the system needs

[1] https://github.com/dos-group/SyncMesh

to be able to cope with and automatically adjust to poor network connectivity without intrusive impact on the local processing of data.

Enable local data processing and storage. In order to not overload the network layer and subsequently interfere with other and possibly critical network traffic, the data needs to be stored, processed, transformed, or aggregated where it is created. Thus, data locality needs to be enabled and the local processing simplified.

Autonomous operation and resilience. Due to the dynamic nature of the aforementioned IoT environments, lightweight edge nodes are expected to join and leave the network at any time. Therefore, the system needs to cope automatically with node joins, churns, and failures by, i.e., enabling the entities to work as autonomous as possible.

Environmental awareness. Although the devices are expected to work in an autonomous manner, to increase the remote sensing capabilities and subsequently the environmental awareness, nodes need to be able to exchange their raw or aggregated data on-demand with other nodes in the proximity.

3.3 System Architecture

With the given assumptions on the environment and subsequently to fit the requirements stated before, we implemented a prototype based on lightweight containerization technologies, which improves the data locality and simplifies the deployment or workload by utilizing FaaS capabilities.

Therefore, each device in the environment – in the following called a *SyncMesh node* – acts as an autonomous node in the mesh network, stores sensor data, and offers an interface that enables users and other nodes in the proximity to collaborate with it.

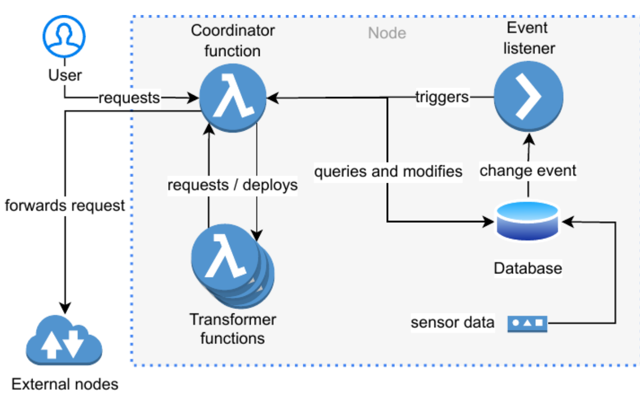


Figure 1: Architecture and key components of a SyncMesh node.

A *SyncMesh node* contains several key components, which will be discussed in more detail in the remainder of this section.

Local database. Each node employs a local database which is used to store data generated by connected sensors and the availability model of other available devices in the proximity. The database can further be accessed by the coordinator function and other available functions on the device and offers the possibility to use event listeners to, i.e., automatically react to changes in the database. In our prototype, we use a MongoDB instance, although other solutions can be used where appropriate.

Database event listener. A separately instantiated database listener registers new sensor data input and forwards the change events to the coordinator function. Therefore, it provides notification functionalities, sharing database changes to subscribed external nodes or calling (transformer) functions for additional processing.

Coordinator function. The *SyncMesh* interface is exposed via the coordinator function and is used as an entry point for users and other nodes in the sensor network to access data on the respective node. In general, it is a serverless function implemented in the OpenFaaS framework and provided via an HTTP API. Every request uses the same unified request schema containing the data query as a GraphQL statement, as well as additional parameters like for instance transformer functions.

The coordinator function can store and modify data on the node itself by interacting with the local database. In addition, it forwards events registered by the database listener to subscribed nodes, enabling decentralized data replication as well as event-driven mechanisms. Although the coordinator function already implements aggregation functionalities, it forwards more advanced data preprocessing tasks to other local transformer functions that enable diverse workloads, from simple analysis to Machine Learning (ML) workloads. Finally, before sending the data to the client, the response is compressed with gzip.

Transformer functions. In order to reduce the data transmission between nodes and to facilitate operation in limited bandwidth scenarios, transformer functions can be used to pre-process data directly on the respective nodes. Sensor data can, i.e., be aggregated and analyzed with ML models, e.g., image classifiers or other data analysis pipelines, before being transferred over the network.

The envisioned workload is implemented in a function or a chain of functions, provided in docker containers and started by the coordinator function in case of changes in the database or when triggered externally, e.g. by nodes in the proximity. The respective containers are scaled on-demand as well as stopped in case they are not needed anymore, therefore relieving resources of lightweight edge nodes.

Requirement analysis. In summary, in order to support the operation during unreliable network conditions, *SyncMesh* nodes only exchange data on-demand and favor local data pre-processing as much as possible, subsequently relieving the network layer. The sensor data is initially only stored on the respective devices, and by leveraging FaaS capabilities, the local processing is simplified via the deployment of arbitrary (transformer) functions. Due to the stateless nature of the serverless functions, they can be scaled up and down if needed, thus automatically adapting to increasing and decreasing demand. Therefore, a single *SyncMesh* node acts autonomously in regard to processing the locally available (sensor) data.

Further, the data can also be exchanged and replicated across other available nodes in the environment on-demand. For that reason, each *SyncMesh* node maintains a model of other available devices in the proximity [2] and transparently forwards user requests regarding sensor readings in the surrounding area to respective neighboring nodes, possibly aggregating the results and returning a unified response. The requests are only forwarded to currently available nodes. Consequently, *SyncMesh* is automatically adjusting to node churns and joins.

Finally, since all components and used functions in *Syncmesh* are provided as containers, they can be adapted to heterogeneous CPU architectures by i.e. providing multi-arch Docker images [2].

4 EVALUATION

In order to evaluate our *SyncMesh* prototype, we conducted a set of experiments, comparing it to several baselines and describe our results in this section.

4.1 Experiment setup

To simulate an environment as described in our assumptions, we created a virtual testbed on the Google Cloud Platform. The hardware and software specifications for each used virtual machine instance are listed in Table 1. As depicted in Figure 2, we deployed virtual machines in different cloud regions to introduce latencies between participating nodes in a distributed environment. Subsequently this resulted in latencies ranging between 20 and 300ms. In addition, in some cases we used a separate virtual machine as an external server for the implemented baselines.

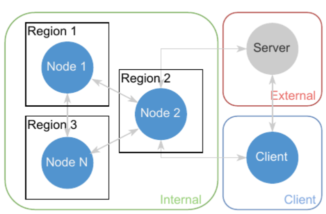


Figure 2: A basic network diagram visualizing the experiment setup.

In order to simulate sensors connected to the edge devices, we leveraged a real-world IoT dataset, containing air quality, temperature and humidity measurements collected in a city in Bulgaria [3]. The dataset was distributed across the nodes and added to the respective databases, simulating real world measurements.

For the evaluation, we considered two different scenarios:

- Collect: A client requests all data from all nodes of a sensor network for a given time span. Therefore, the sensor data is retrieved from all participating nodes and provided to the user.
- Transform: A client requests i.e. aggregated data from the surroundings of an edge node. Therefore, the sensor data is, if applicable, preprocessed and then provided to the user.

As illustrated in Figure 2, we monitored the used traffic for requests from the client (blue), internal network traffic between the nodes (green), and in case of baselines from and to the external cloud server (red). In addition, we used the Round Trip Time (RTT) until a clients request was successfully processed as a further evaluation metric.

Finally, we performed the experiments for different amounts of nodes in the network (3, 6, 9 and 12) and several data time spans (1, 7, 14 and 30 days). Each configuration was repeated 20 times to improve accuracy and soften outliers.

The used scripts, software and additional tooling are provided in our GitHub repository.

Table 1: Hard and Software Specifications

Hardware (each GCP VM)	
CPU	1vCPU @ 2.60GHz
Memory	3.75 GB
Storage	20 GB
Software	
Ubuntu	20.04-lts
MongoDB	5.0.2
GUN	0.2020.1235
Node	14
faasd	0.14.2

4.2 Baselines

We implemented three other systems to compare them against *SyncMesh*. These baselines were chosen because they are being used in practice to handle vast amounts of data in IoT networks.

central cloud storage
: A standard implementation of a sensor network. Each sensor sends the data to a central cloud database (MongoDB) from where clients can retrieve it.

sharded cloud storage
: A sharded implementation of the central cloud storage. Each sensor stores its own data, and a central server is supplying unified access to the client. Each sensor thereby represents a shard in the database.

p2p distributed database storage
: A peer-to-peer database with eventual consistency based on GunDB[4]. Clients need to individually connect to peers in the network in order to retrieve all sensor data or synchronize data between nodes.

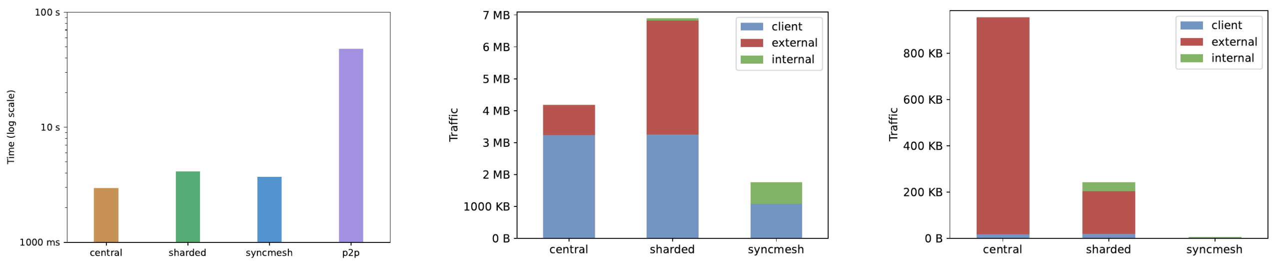
In contrast to the other baselines, p2p sends all data uncompressed. The MongoDB based systems use snappy and *SyncMesh* implements a gzip compression.

4.3 Results

In this section we first discuss the measurements of our comparison benchmark.

Figure 3 describes the evaluation results of a three node sensor network. Therefore, Figure 3a depicts the time a client needs to retrieve all sensor data from all nodes, when using different baselines

[2] https://docs.docker.com/desktop/multi-arch/
[3] https://airsofia.info/
[4] https://gun.eco/

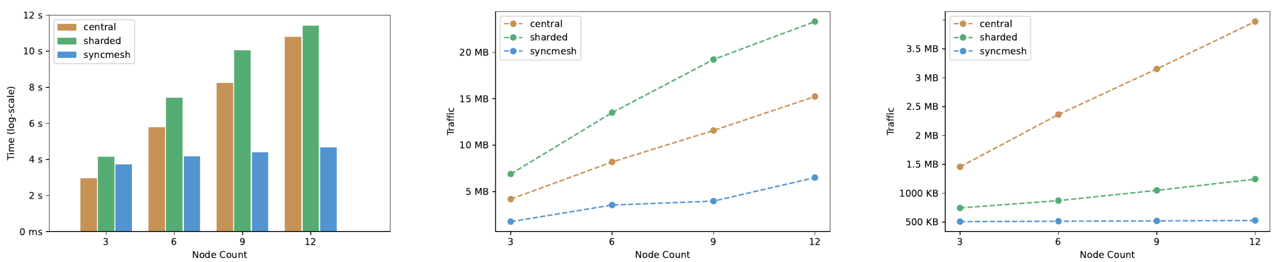


(a) Time until all sensor data of a three node cluster is retrieved by the client, including the results of the p2p baseline. The y-axis is log scaled to enable the representation in a single plot.

(b) Distribution of network traffic for the collect scenario on different systems in a sensor network consisting of three nodes.

(c) Distribution of network traffic for the transform scenario (average sensor readings) in a sensor network consisting of three nodes.

Figure 3: Exemplary request time for a three node network including the p2p baseline (a) and the distribution of monitored network traffic between nodes in the network, the client and respectively a central server during the collect (b) and transform (c) scenario.



(a) Time needed to retrieve all data from a sensor network, for different baselines, network sizes and the collect scenario (30 days). For the baselines, the graph also includes the time needed to send the data to i.e. the central database.

(b) Used traffic in the sensor network for different baselines, network sizes and the collect scenario (30 days).

(c) Used traffic in the sensor network for different baselines, network sizes and the transform scenario (average sensor readings of last 30 days).

Figure 4: Request time (a) and combined traffic between nodes, the client and respectively external server for the collect (b) and transform (c) scenarios in different network sizes.

and *SyncMesh*. As can be seen, the *p2p* baseline needs significant more time than all other baselines, which is why we excluded it in other plots to improve the detectability. Furthermore, *SyncMesh* performs 10.1% better than *sharded* and 20.7% worse than the *central* baseline.

In Figure 3b we show the produced traffic for each systems during the collect scenario, except for the *p2p* baseline (91.03 MB). The results further show that *SyncMesh* is able to outperform the *sharded* and *central* baselines by up to 57.8% in regard to network efficiency. This is in part due to the different compression algorithms used in MongoDB and *SyncMesh*. In the case of *p2p* the amount of data sent is even doubled, due to protocol specifics of the employed solution. In addition, *SyncMesh* therefore also produces less internal traffic between participating nodes than all of the other solutions.

This is especially evident in Figure 3c, which depicts the used traffic during the transform scenario. *SyncMesh* is again able to outperform the baselines by only using 6.99 kB. As can be seen, the majority of used traffic results from transmitting the sensor data to the central server. In contrast to Figure 3c, the sharded system performs 75.0% better than the central system, as the sharded MongoDB also uses local workers (for aggregation) on each shard, decreasing the amount of traffic between the server and its shards.

The plots in Figure 4 describe the same experiment as Figure 3, but for a three, six, nine and twelve node network respectively. As expected and depicted in Figure 3a, the *sharded* and *central* baseline show a continuous increase in request times for growing network sizes due to the increased data amount that must be send to the cloud from all nodes and retrieved by the client. However, the performance of *SyncMesh* does not degrade significantly: The time is only increased by 25.7%, in comparison to 175.6% for *shared* and 266.1% in case of *central*, most likely due to the decreased traffic between nodes.

This trend is also observable in Figure 4b and Figure 4c, that describe the used traffic for the collect and transform scenarios across different network sizes. As can be seen, *SyncMesh* scales significantly better in regard to transmitted traffic, and therefore is able to exonerate the network link between participating nodes in the network.

5 CONCLUSION

In this paper we have introduced *SyncMesh*, a system that improves the data locality and enables local data processing based on the FaaS paradigm in a meshed network of edge nodes. We prototypical implemented *SyncMesh* and evaluated our approach in a virtualized testbed on a real world IoT dataset, and compared it against several baselines in regard to request time and network utilization.

As indicated by the preliminary results, *SyncMesh* is able to outperform the traditional approaches in regard to network usage and request times, in particular for larger amounts of nodes.

Consequently, our results underline the importance and impact of data locality, especially for the the application of FaaS in highly distributed environments.

In future work we plan to further extend the *SyncMesh* prototype, include more complex analysis workloads and evaluate our approach on real-world edge devices. In addition, we intend to assess the impact of cold starts in the FaaS framework.

ACKNOWLEDGMENTS

We would like to thank Denis Koljada for contributing to the initial prototype within the scope of a master's project.

REFERENCES

[1] Eyuel D Ayele, Nirvana Meratnia, and Paul JM Havinga. 2018. Towards a new opportunistic iot network architecture for wildlife monitoring system. In *NTMS*. IEEE.

[2] Soeren Becker, Florian Schmidt, Lauritz Thamsen, Ana Juan Ferrer, and Odej Kao. 2021. LOS: Local-Optimistic Scheduling of Periodic Model Training For Anomaly Detection on Sensor Data Streams in Meshed Edge Networks. In *ACSOS*. IEEE.

[3] David Bermbach, Jonathan Bader, Jonathan Hasenburg, Tobias Pfandzelter, and Lauritz Thamsen. 2021. AuctionWhisk: Using an auction-inspired approach for function placement in serverless fog platforms. *Software: Practice and Experience* (2021).

[4] Josiah Burchard, Dmitrii Chemodanov, John Gillis, and Prasad Calyam. 2017. Wireless Mesh networking Protocol for sustained throughput in edge computing. In *ICNC*. IEEE.

[5] Bin Cheng, Jonathan Fuerst, Gurkan Solmaz, and Takuya Sanada. 2019. Fog function: Serverless fog computing for data intensive IoT services. In *SCC*. IEEE.

[6] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. 2018. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet of Things Journal* 5, 2 (2018).

[7] Mung Chiang and Tao Zhang. 2016. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal* 3, 6 (2016).

[8] Bastien Confais, Adrien Lebre, and Benoît Parrein. [n. d.]. An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS. In *ICFEC*. IEEE.

[9] Bastien Confais, Adrien Lebre, and Benoît Parrein. 2016. Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures. In *CloudCom*. IEEE.

[10] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. Serverless applications: Why, when, and how? *IEEE Software* 38, 1 (2020).

[11] Ana Juan Ferrer, Soren Becker, Florian Schmidt, Lauritz Thamsen, and Odej Kao. 2021. Towards a Cognitive Compute Continuum: An Architecture for Ad-Hoc Self-Managed Swarms. In *CCGrid*. IEEE.

[12] Anisha Gupta, Rivana Christie, and PR Manjula. 2017. Scalability in internet of things: features, techniques and research challenges. *Int. J. Comput. Intell. Res* 13, 7 (2017).

[13] Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless computing: One step forward, two steps back. In *CIDR*. ACM.

[14] Yu Liu, Kin-Fai Tong, Xiangdong Qiu, Ying Liu, and Xuyang Ding. 2017. Wireless mesh networks in IoT networks. In *iWEM*. IEEE.

[15] Ruben Mayer, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. 2018. FogStore: Toward a distributed data store for Fog computing. *2017 IEEE Fog World Congress, FWC 2017* (5 2018), 1–6. https://doi.org/10.1109/FWC.2017.8368524

[16] Vasileios Moysiadis, Panagiotis Sarigiannidis, and Ioannis Moscholios. 2018. Towards Distributed Data Management in Fog Computing.

[17] Arslan Munir, Prasanna Kansakar, and Samee U. Khan. 2017. IFCIoT: Integrated Fog Cloud IoT: A novel architectural paradigm for the future Internet of Things. *IEEE Consumer Electronics Magazine* 6, 3 (2017).

[18] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. 2017. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing* 21, 4 (2017).

[19] Tobias Pfandzelter and David Bermbach. 2020. TinyFaaS: A Lightweight FaaS Platform for Edge Environments. In *ICFC*. IEEE.

[20] Pierluigi Plebani, David Garcia-Perez, Maya Anderson, David Bermbach, Cinzia Cappiello, Ronen I. Kat, Frank Pallas, Barbara Pernici, Stefan Tai, and Monica Vitali. 2017. Information logistics and fog computing: The DITAS approach. In *CAiSE*. Springer.

[21] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. 2021. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems* 114 (2021).

[22] Maryam Sheikh Sofla, Mostafa Haghi Kashani, Ebrahim Mahdipour, and Reza Faghih Mirzaee. 2021. Towards effective offloading mechanisms in fog computing. *Multimedia Tools and Applications* (2021).

[23] Weisong Shi and Schahram Dustdar. 2016. The promise of edge computing. *Computer* 49, 5 (2016).

[24] Luiz Angelo Steffenel. 2019. Improving the Performance of Fog Computing Through the Use of Data Locality. In *SBAC-PAD*. IEEE.

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

[26] Tian Wang, Jiyuan Zhou, Anfeng Liu, Md Zakirul Alam Bhuiyan, Guojun Wang, and Weijia Jia. 2019. Fog-based computing and storage offloading for data synchronization in IoT. *IEEE Internet of Things Journal* 6, 3 (2019).

[27] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2019. The nebulastream platform: Data and application management for the internet of things. *arXiv preprint arXiv:1910.07867* (2019).