KuVS-Fachgespräch Fog Computing 2020

Zoltán Ádám Mann, Stefan Schulte (Eds.)

zoltan.mann@paluno.uni-due.de, s.schulte@dsg.tuwien.ac.at

Technical Report Essen, Vienna, April 2020

https://www.kuvs.de/fg/fogcomputing/

Editors

Zoltán Ádám Mann
University of Duisburg-Essen
paluno – The Ruhr Institute for Software Technology
Gerlingstr. 16
45127 Essen
Germany
zoltan.mann@paluno.uni-due.de

Stefan Schulte
TU Wien, Distributed Systems Group
Argentinierstrasse 8/194-2
1040 Vienna
Austria
s.schulte@dsg.tuwien.ac.at

Copyright © 2020 for the individual papers by the papers' authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors.

PREFACE

The "Fachgespräche" series of the *Communication and Distributed Systems* (German: *Kommunikation und Verteilte Systeme* – KuVS) special interest group[1] of the German Society for Computer Science (GI) and the Information Technology Society (ITG) of VDE brings together researchers in order to discuss fresh ideas in recent research topics. As such, we are very grateful that it was possible to organize another edition of the GI/ITG KuVS-Fachgespräch Fog Computing[2] as part of this series.

Since the inaugural edition of the KuVS-Fachgespräch Fog Computing in 2018, the topic has evolved to a major research direction in the field of distributed systems, with different conferences and journals focusing on fog and edge computing. We are happy that the 2nd GI/ITG KuVS-Fachgespräch Fog Computing received a substantial number of submissions, both from research groups already involved in the inaugural edition, and from groups new to this Fachgespräch.

Overall, seven papers were accepted for presentation. Each paper received at least three reviews by the Program Committee members.

In their paper *Survey of Mobile Opportunistic Networks for Parallel Data Dissemination and Processing*, Peter Danielis and Gunnar Karlsson discuss approaches to opportunistically disseminate and process data in mobile networks, including approaches which make use of fog computing. Jonathan Hasenburg and David Bermbach present a novel approach to identify the relevance of IoT data for broker-based publish/subscribe in their paper *Using Geo-context Information for Efficient Rendezvous-based Routing in Publish/Subscribe Systems*. Alexander Palm et al. investigate in their work *Towards Online Reinforcement Learning for Self-adaptive Fog Systems* the utilization of reinforcement learning for the adaptation of applications which exploit fog resources. Vasileios Karagiannis examines how new fog nodes can be added to an existing fog environment in his paper *Building A Scalable Distributed System For Fog Computing*. In their paper *Fogsy: Towards Holistic Industrial AI Management in Fog and Edge Environments*, Patrick Wiener et al. discuss how Artificial Intelligence can be realized in the fog and at the edge. Simon Krejci and Stefan Schulte investigate which typical edge devices can be used in order to interact with the InterPlanetary File System and the Ethereum blockchain in their paper *Living at the Edge: Running IPFS and Ethereum Client Software on Single-board Computers*. Finally, Julian Bellendorf studies in his paper *Latency in Fog Computing* which different types and sub-types of latency play a role in the fog.

In order to minimize the risk for the participants of the Fachgespräch, it was decided to postpone the actual on-site meeting to a later date in 2020, i.e., to a date when the Covid-19-induced travel restrictions in Germany have been lifted. Therefore, these proceedings are actually pre-proceedings to the later on-site meeting in Essen, Germany.

Last but not least, the organizers would like to thank everybody involved in the organization of the 2nd GI/ITG KuVS-Fachgespräch Fog Computing, especially the local organizers at the University of Duisburg-Essen and the Program Committee members:

- Atakan Aral, TU Wien
- Peter Danielis, Universität Rostock
- Frank Dürr, Universität Stuttgart
- Boris Koldehofe, Rijksuniversiteit Groningen
- Ruben Mayer, TU München
- Andreas Reinhardt, TU Clausthal
- Dominik Riemer, FZI Forschungszentrum Informatik, Karlsruhe
- Lin Wang, VU Amsterdam

Without their help, it would not have been possible to organize the Fachgespräch. Also, we thank our sponsor Ascora[3] for supporting this event.

Essen, Vienna, April 17, 2020

Zoltán Ádám Mann & Stefan Schulte

[1] https://www.kuvs.de/

[2] https://sites.google.com/view/fachgespraechfog/

[3] https://ascora.net/

CONTENTS

Survey of Mobile Opportunistic Networks for Parallel Data Dissemination and Processing

Peter Danielis
Department of Computer Science
University of Rostock
Rostock, Germany
peter.danielis@uni-rostock.de

Gunnar Karlsson
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
gk@kth.se

***Abstract*—In this paper, we survey work that relates to our fully distributed protocol called UrbanCount for counting large numbers of people using opportunistic device-to-device communication. We provide an overview of existing approaches that address mobile opportunistic networks for parallel data dissemination and processing. First, works are presented that are concerned exclusively with the dissemination of information in opportunistic networks. We continue to present approaches that are dedicated to both opportunistic data dissemination and processing in mobile networks. In this context, we outline gossiping algorithms and fog computing-based techniques. Subsequently, we present methods that use mobile crowd sensing for data dissemination and processing. We contrast them with procedures that leverage opportunistic data processing and finally we conclude open research questions. In this regard, our research objectives are to investigate how the modeling can be carried out for a scenario such as UrbanCount and for which other use cases opportunistic networks for data dissemination and processing are suitable.**

***Index Terms*—mobile opportunistic networks, parallel data dissemination, parallel data processing**

I. Introduction

In an opportunistic network, nodes can exchange information, for example, by using device-to-device communication once they are in direct communication range of each other [1]. In mobile opportunistic networks, the underlying node mobility causes nodes to meet occasionally, thereby helping to disseminate data. For instance, a node can be a person carrying a device, such as a cell phone, that is equipped with a wireless communication interface [2]. In the context of parallel computing, each node participating in the opportunistic network acts as a collector and processor of data and as a disseminator of the result of the calculation. The calculation is often carried out iteratively between the nodes.

Parallel computing in opportunistic networks is not always easy, especially when it comes to open systems with node churn, i.e., nodes that enter and leave the area under consideration. As an example for parallel computing in open systems with mobile nodes, we evaluated the performance of a distributed node counting protocol called UrbanCount using simulations [3]. Each UrbanCount node collects estimates of the people count from other participants in the system whenever in direct communication range and immediately integrates these estimates into a local estimate. We showed that the accuracy of the counting measure strongly depends on the node density. In open systems such as those under investigation in [3], the counting must also consider nodes that leave the area during the period for which the count is being performed. In closed systems, on the contrary, the number of nodes is fixed, but unknown and must be estimated. One use case for closed systems is to estimate attendance at an event where churn is low and can possibly be neglected. This defines exactly when the parallel calculations are completed: when all interested nodes in the system have calculated the same value.

In this paper, we survey work that relates to our work UrbanCount and finally we formulate as a research question how the modeling could be carried out for an open system like UrbanCount. Furthermore, we raise the question of which tasks, for which the calculation is more difficult than for UrbanCount, could also be solved by parallel data distribution and processing in opportunistic networks. A brief outlook shows initial ideas for answering these questions.

II. State of the Art

There is a number of works in the literature that are concerned with the dissemination of information in opportunistic networks. Such networks can, for example, help to increase the resilience of communication systems to disasters [4]. As described in [4], mobile phones can be organized into a peer-to-peer network for this purpose. Networking is made possible by device-to-device communication as soon as devices are in direct transmission range of each other. As a result, the devices are not dependent on the connection to cellular towers, which can possibly not be established in the event of a disaster. A model that maps the properties of opportunistic networks is described in [5]. With the model, the authors pursue the goal of realistically representing the spread of information. The work in [6] presents an analytical model based on population processes. This model serves to characterize the opportunistic information dissemination in 5G networks. The authors present closed-form expressions that can be used to determine the diffusion time, network coverage, and latency. Another analytical model is developed in [7]. It models the dependencies between network-wide inter-contact time and the inter-contact time per network node. Another work from the state of the art addresses the possibilities

that mobile opportunistic networks offer to spread time and space sensitive information [8]. The study in [9] examines opportunistic information dissemination, which serves to store mobile data in an open system. All of the work addressed so far, however, have in common that they solely consider the exchange of information between network nodes. Hence, they do not use the full scope of functions available in opportunistic networks.

Opportunistic networks are used for data exchange, but data processing is also possible. Data exchange is solely possible during the sporadic contact between nodes, which occurs as soon as two or more nodes are within within communication range of each other [10]. In addition to the opportunistic data exchange, the nodes can use their respective software and hardware resources to work on tasks in parallel even outside the sporadically occurring contacts. One such task can be to calculate the total value over the entire network from the local values of a node. In this context, there are numerous gossiping algorithms that calculate a value across the network, such as an average or a sum of a given metric. Nodes usually exchange data periodically in synchronous rounds. Typically, only randomly selected or predefined nodes work during a round. The goal is for all nodes to calculate a common value [11]. Gossiping algorithms are even able to work in networks that are characterized by regular node entry and exit, i.e, churn.

A gossiping algorithm for mobile networks is presented in [12], in which network nodes only know a few neighbors. A fully distributed gossip-based protocol for aggregating values in large dynamic networks is described in [13]. By means of paired interaction between network nodes, the locally aggregated values of all nodes quickly converge to the global aggregation value. The work in [14] investigates the data mining of data collected by edge devices such as mobile phones. A gossip-learning approach is used for reasons of privacy and robustness. Using this approach, models of fully distributed data can be learned in large-scale networks. Each node in the network has exactly one data record, such as a sensor reading, which it does not share with other nodes. The gossip-learning algorithm includes models that perform random walks across all nodes in the network, and the models are updated each time they visit a node by using the local data record. There are as many models as nodes in the network due to the procedure. Since models perform random walks, all nodes are exposed to a continuous stream of models that pass through them. In addition to using these models directly for prediction, nodes can also combine them in different ways using ensemble learning.

Opportunistic processing differs from gossiping algorithms in particular in that parallel pairs of nodes communicate simultaneously. Neighborhood lists are not known in advance and not all known neighboring nodes are requested before the connection to one of them is established. This is often due to the dynamics of the system, which occur faster than what is usually the case in studies of gossiping algorithms. The typical link duration in opportunistic networks with pedestrians is typically 10 s [15].

Mobile and fog computing have developed to bring computing power closer to the end user. This can be observed particularly in connection with large IoT networks. In principle, the fog includes computing units that have sufficient storage, energy, and computing resources and are able to carry out parallel information processing [16]. Some work extends the fog to include end-user devices such as mobile devices or sensors. An approach to resource sharing in very dense IoT mesh networks is presented in [17]. This method combines data communication and processing. By implementing an artificial neural network on top of an IoT network, communication between users can be used for data aggregation and processing. At the same time, energy efficiency is increased and the latency for processing is reduced. However, the scenario examined does not include any mobile nodes and is based on a closed system with predefined functionality.

To use the resources on mobile devices for data collection and processing, mobile crowd sensing is one solution [18]. An opportunistic variant of this approach is explained in [19]. The method presented collects data in the background without the user becoming active. Mobile crowd-sensing applications typically use the client-server network model and so the data collected by the devices is sent to a central server. It carries out the further data processing and distributes the results to interested users. Another work uses peer-to-peer approaches, in which, however, only the information exchange takes place directly between the nodes [20]. The information processing is still carried out centrally on a server. If one compares mobile crowd sensing with opportunistic data processing, it can be seen that the nodes in the latter variant both collect and process data. Finally, they distribute results of the data processing to other nodes.

The work in [21] gives an overview of various non-image-based methods for counting people. One of these techniques is based on opportunistic data exchange using device-to-device communication in mobile networks with churn [3]. This technique is supplemented by an exploratory study in [22]. This study deals with the simulation of closed systems to investigate the interaction between opportunistic communication and parallel data processing on the nodes. Another work [23] examines the use of opportunistic communication by means of device-to-device communication for distributed voting, again in mobile networks with churn.

III. Conclusion and Research Questions

To conclude, let us focus on our proposals, which deal with both data dissemination and processing using device-to-device communication in mobile opportunistic networks for the use cases of voting and people counting [3], [23]. These works already show that the algorithms presented are suitable for use in highly dynamic mobile opportunistic networks. Compared to state-of-the-art algorithms, the local values of the individual nodes converge to the global value (and also very quickly). However, only the basic process has been designed and simulated but not modeled for the open systems

under investigation. The basic process has been shown to be applicable for the use cases of distributed voting and node counting. It consists of collecting data, performing a calculation of the data, and sharing the result of the calculation in parallel. For the already studied use cases, the process is contact limited (limited by the contact duration) because the calculation is only done when new information is provided in a node (and for counting, the processing is light). Other parallel opportunistic calculations may have heavier calculations and be less dependent on the exchange of intermediate results with other nodes in the system. Data in the previously investigated uses case are basically node IDs, but could have been other system parameters that should be collectively estimated, or data could be environmental parameters for the space that the nodes jointly detect, collect, and calculate. Hence, the previous results may not be applicable in other cases.

Based on these insights and experiences, let us conclude following research questions:

- How to develop a model that captures the dynamics in open systems characterized by node churn?
- Which tasks other than counting and voting could be solved through parallel data dissemination and processing by mobile opportunistic communication?

Therefore, our first research objective is to develop a model that maps the dynamics in scenarios like UrbanCount. An initial model for mapping the process of collecting data opportunistically from a closed set of nodes has been developed in [22] but is only valid for closed systems. Hence, this is solely a first step towards a model that captures the dynamics in open systems with churn, but, however, in the closed systems we were able to show that the process can be well described by empirical laws.

Another task that could be accomplished by parallel data exchange and processing is the monitoring of critical infrastructures by a swarm of autonomous robots both under water and in the air. The robots are constrained in terms of communication range and energy and should be coordinated as well as possible to perform the task, on which they work in parallel. As opposed to our previous works, in which we investigate the impact of given movement patterns, we will rather determine the best movement behavior under the given constraints in order to complete the task in a certain time.

REFERENCES

[1] T. Li, Z. Xiao, H. M. Georges, Z. Luo, and D. Wang, "Performance analysis of co-and cross-tier device-to-device communication underlaying macro-small cell wireless networks." *KSII Transactions on Internet & Information Systems*, vol. 10, no. 4, 2016.

[2] Ó. Helgason, S. T. Kouyoumdjieva, L. Pajević, E. A. Yavuz, and G. Karlsson, "A middleware for opportunistic content distribution," *Computer Networks*, vol. 107, pp. 178–193, 2016.

[3] P. Danielis, S. T. Kouyoumdjieva, and G. Karlsson, "Urbancount: Mobile crowd counting in urban environments," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2017, pp. 640–648.

[4] P. J. Denning and D. Brin, "An interview with David Brin on resiliency," *Communications of the ACM*, vol. 62, no. 6, pp. 28–31, 2019.

[5] Z. Vatandas, S. M. Hamm, K. Kuladinithi, U. Killat, A. Timm-Giel, and A. Förster, "Modeling of data dissemination in oppnets," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2018, pp. 01–04.

[6] E. Hernández-Orallo, M. Murillo-Arcila, J.-C. Cano, C. T. Calafate, J. A. Conejero, and P. Manzoni, "An analytical model based on population processes to characterize data dissemination in 5g opportunistic networks," *IEEE Access*, vol. 6, pp. 1603–1615, 2017.

[7] A. Passarella and M. Conti, "Characterising aggregate inter-contact times in heterogeneous opportunistic networks," in *International Conference on Research in Networking*. Springer, 2011, pp. 301–313.

[8] S. Wang, X. Wang, J. Huang, R. Bie, Z. Tian, and F. Zhao, "The potential of mobile opportunistic networks for data disseminations," *IEEE Transactions on vehicular Technology*, vol. 65, no. 2, pp. 912–922, 2015.

[9] S. T. Kouyoumdjieva and G. Karlsson, "Energy-aware opportunistic mobile data offloading under full and limited cooperation," *Computer Communications*, vol. 84, pp. 84–95, 2016.

[10] M. Conti, S. Giordano, M. May, and A. Passarella, "From opportunistic networks to opportunistic computing," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 126–139, 2010.

[11] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[12] Z. Shi and P. K. Srimani, "An online distributed gossiping protocol for mobile networks," *Journal of combinatorial optimization*, vol. 11, no. 1, pp. 87–97, 2006.

[13] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[14] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[15] Ó. Helgason, S. T. Kouyoumdjieva, and G. Karlsson, "Opportunistic communication and human mobility," *IEEE Transactions on Mobile Computing*, vol. 13, no. 7, pp. 1597–1610, 2013.

[16] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[17] E. Di Pascale, I. Macaluso, A. Nag, M. Kelly, and L. Doyle, "The network as a computer: A framework for distributed computing over iot mesh networks," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2107–2119, 2018.

[18] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[19] H. Ma, D. Zhao, and P. Yuan, "Opportunities in mobile crowd sensing," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 29–35, 2014.

[20] C. Jiang, L. Gao, L. Duan, and J. Huang, "Scalable mobile crowdsensing via peer-to-peer data sharing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 4, pp. 898–912, 2017.

[21] S. T. Kouyoumdjieva, P. Danielis, and G. Karlsson, "Survey of non-image based approaches for counting people," *IEEE Communications Surveys & Tutorials*, 2019.

[22] T. Li, S. T. Kouyoumdjieva, G. Karlsson, and P. Hui, "Data collection and node counting by opportunistic communication," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019, pp. 1–9.

[23] P. Danielis, S. T. Kouyoumdjieva, and G. Karlsson, "Divote: A distributed voting protocol for mobile device-to-device communication," in *2016 28th International Teletraffic Congress (ITC 28)*, vol. 1. IEEE, 2016, pp. 69–77.

Using geo-context information for efficient rendezvous-based routing in publish/subscribe systems

Jonathan Hasenburg, David Bermbach
TU Berlin & Einstein Center Digital Future
Mobile Cloud Computing Research Group
{jh, db}@mcc.tu-berlin.de

***Abstract*—A promising communication paradigm that enables communication between IoT devices is broker-based publish/subscribe. When the brokers are distributed across the fog, events and subscriptions of clients connected to different broker instances must be routed across the router network. State-of-the art solutions, however, do not take into account that the relevance of the IoT data depends on its origin and purpose. Instead, they assume a uniform data distribution when determining where to match events and subscriptions which degrades system performance.**

In this paper, we propose a routing solution that builds upon geo-context information attached to published events and subscriptions. This way, we can match events close to either the publishers or subscribers of an event, thus, minimizing communication latency while not affecting scalability.

***Index Terms*—Geo-Context, IoT Data, Geo-Distributed Pub/Sub**

I. Introduction

The vision of the Internet of Things (IoT) is to connect billions of devices. These devices usually operate at the edge of the network; thus, they might be battery powered or have a slow and unstable connection to the wide area network. Hence, rather than interconnecting devices directly, communication is usually handled by some kind of distributed middleware operating in the fog. A promising communication paradigm for this purpose is broker-based publish/subscribe (pub/sub) because it allows client devices to asynchronously communicate without having to know each other [1]: they create subscriptions (subscribers) and send events (publishers) to any of the brokers, brokers match incoming events with created subscriptions and deliver them accordingly.

Because a large chunk of IoT data is only relevant to a relatively small amount of subscribers [2], which are often operating in physical proximity to each other, communication can often be handled by a local fog broker in the same region. Other use cases, however, require inter-region communication, so brokers must be connected to exchange events and/or subscriptions they received from client devices. How this can be achieved for IoT data continues to be an open research question as existing approaches do not take into account that the relevance of data depends on the data origin (i.e., the current location of the sensor) and purpose (e.g., collecting temperature values to control heaters in proximity or to collecting weather information for nation wide forecasts).

In general, existing pub/sub routing strategies can be classified into the three categories *flooding*, *gossipping*, and *selective* [3], [4]. Flooding does not scale as here every broker has to process all events or subscriptions from every other broker. Gossiping sacrifices latency in favor of tolerance of very dynamic environments by distributing messages between brokers randomly. While the IoT devices might operate in such an environment, the brokers, to which the routing approach is applied, do not. Instead, it is more likely that the brokers are deployed in (a limited number of) fog regions which do not face a high churn rate. Selective approaches are either filter-based or build upon rendezvous points (RP). For the former, filters are distributed across brokers and used to build dynamic multicast trees for each event. Traversing the multicast trees, however, increases end-to-end latency. RPs are are effective in reducing excess data by being a "meeting point" for subscriptions and events for the matching to occur; however, state of the art solutions expect a uniform distribution of data, i.e., they do not take into account that IoT data is often only relevant in a very specific area.

In our previous research [5], [6], we showed that enriching IoT events and subscriptions with geo-context information can reduce excess data and enable new application scenarios. In this paper, we propose to make additional use of this geo-context information to select RPs to ensure data is matched in proximity to where it is relevant which improves system performance. For that sake, we:

- describe how to enrich events and subscriptions with geo-context information (Section II),
- introduce our novel approach on how to use this geo-context information to select appropriate RPs (Section III).

Finally, we draw a conclusion and outline next steps (Section IV).

II. Enriching events and subscriptions with geo-contexts

Before we explain how to use geo-context information to select RPs, we repeat our previous definition of geo-contexts presented in [5], [6]. Note, that we use a slightly updated terminology, as indicated below, that better fits the intended purpose of this paper.

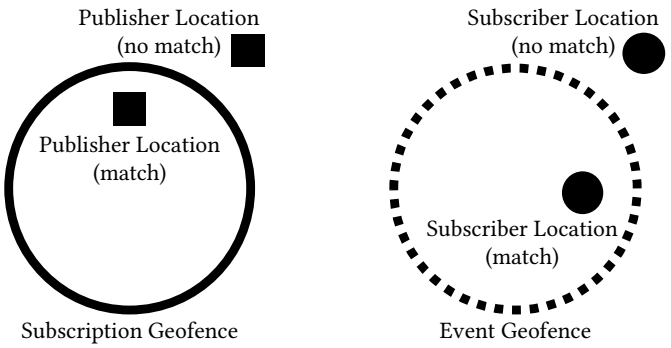


Figure 1. Subscription GeoCheck (left) and event GeoCheck (right) [5], [6].

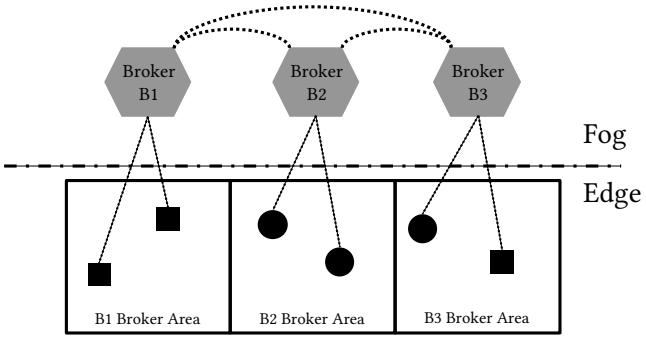


Figure 2. Setup with three brokers deployed in the fog that facilitate communication between publishers (squares) and subscribers (circles).

There are four geo-context dimensions. Clients have a geographic location, which consists of a latitude and a longitude value. For publishers, the corresponding dimension is called **publisher location**; for subscribers, the corresponding dimension is called **subscriber location**. Beyond this, each event and subscription has an area it relates to; we propose to use geofences to describe these areas. The **event geofence**, ensures that only subscribers located in the specified area receive the event, i.e., subscriber locations must be inside the event geofence. The **subscription geofence**, ensures that only the events of publishers located in the specified area may be delivered to the subscriber, i.e., publisher locations must be inside the subscription geofence[1].

For bringing geofences and locations together, two checks are necessary to decide whether data from a given publisher should be delivered to a given subscriber (Figure 1) – first, from the subscribers's perspective with the help of the subscription geofence and publisher locations (subscription GeoCheck) and, second, from the publishers's perspective with the help of the event geofence and subscriber locations (event GeoCheck). For a more detailed discussion and explanation of the geo-context model, we refer to our previous work [5]. Furthermore, it usually makes sense to combine the two GeoChecks with an additional ContentCheck, i.e., based on topics. For a more detailed discussion on how this can be used to build a (single-node) data distribution service leveraging geo-contexts, we refer to our previous work [6]. In this work, we describe how such single-node service instances (here called brokers) can communicate via rendezvous-based routing.

III. Rendezvous node selection

In this section, we present our RP selection approach that builds upon geo-context information. RPs reduce communication cost by being a "meeting point" for subscriptions and events: the matching occurs at the RP brokers [4, p. 166]. Hence, they constrain propagation of events and/or subscriptions to a small subset of nodes which improves system efficiency. A major challenge with rendezvous-based routing in wide-area deployments is to select an RP that is close to the subscribers or publishers of an event. Many state of the art solutions distribute RPs uniformly over available brokers [7], [8]; this is problematic for IoT data traffic which is non-uniform. Our key idea is to use the IoT data itself to identify RPs physically close to publishers and/or subscribers; this is only possible if the necessary geo-context information is attached to the events and subscriptions.

In the following, we first describe some assumptions (Section III-A) before we present how geo-context information can be used to select RPs close to subscribers (Section III-B) or close to publishers (Section III-C). Both strategies come with their own advantages and disadvantages; which one is better depends on the application scenario. Finally, we discuss how both strategies compare against the baseline, flooding of either events or subscriptions to all brokers, and other rendezvous-based routing approaches found in the literature (Section III-D).

A. Assumptions

For our approach, we assume a setup that comprises multiple geo-distributed brokers and clients, i.e., IoT devices and services. Even though brokers are geo-distributed, they are aware of each other, typically have a good inter-connection, and are well equipped in terms of computing power; clients, on the other hand, might operate in a constrained environment and only communicate with a single broker, i.e., their local broker (LB). A broker is responsible for communication with all clients located in its *broker area*; usually, a broker area covers the region surrounding the physical location of the corresponding broker as this asserts low latency communication between the broker and clients located in the area[2], see also Figure 2. Subscriptions and published events comprise the payload, some kind of content filter (e.g., a topic), and geo-context information. When a client creates a subscription, it creates the subscription at its LB. Similarly, when a client publishes an event, it sends the event to its LB. Depending on the strategy (Section III-B and III-C), as soon as the LB has

[1] In previous work we referred these four dimensions as producer location, consumer location, producer geofence and consumer geofence.

[2] Using the network distance instead of physical distance to determine broker areas might be more accurate, but is also more complicated in an environment with changing network conditions.

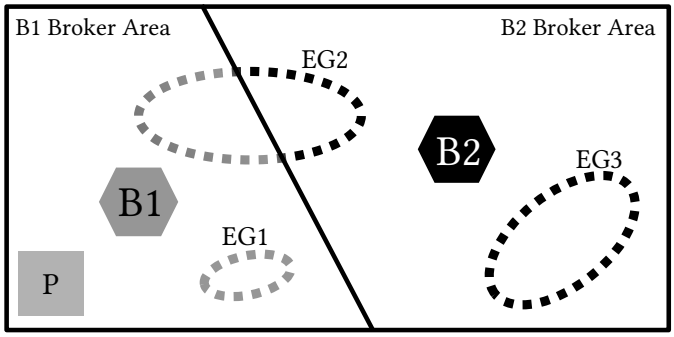


Figure 3. An event only needs to be sent to brokers with a broker area that intersects with the event geofence.

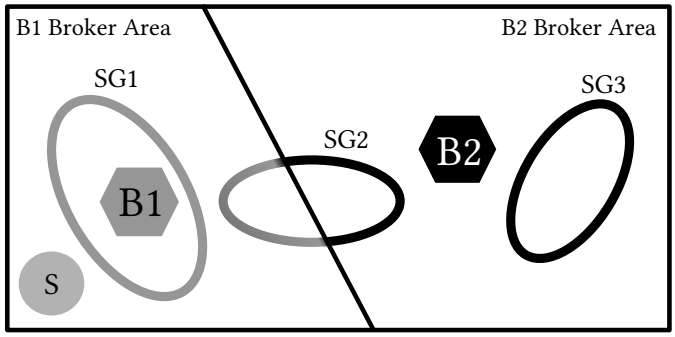


Figure 4. A subscription only needs to be sent to brokers with a broker area that intersects with the subscription geofence.

received an event or subscription, it distributes them to the RP where the matching occurs.

B. Selecting RPs close to the subscribers

With this strategy, the RPs for an event are all brokers that are the respectively closest broker to each of the subscribers that have created a matching subscription. Thus, the RPs are the LBs of these subscribers. Hence, subscriptions are not distributed to other brokers as subscribers create subscriptions at their LB. The event, on the other hand, is distributed to all brokers which could possible manage a matching subscription. Fortunately, the event geofence can be used to select these RP because only broker areas intersecting with the event geofence might contain clients that pass the event GeoCheck (subscriber location inside event geofence).

Figure 3 shows an example with one publisher (P) that is located in the broker area of broker B1 and publishes three events—each has a different event geofence (EG):

- EG1 does not intersect with the broker area of broker B2 so the event does not need to be forwarded for matching to B2.
- EG2 intersects with the broker areas of B1 and B2 so the event needs to be matched at B1 and be forwarded for matching to B2.
- EG3 only intersects with the broker area of B2 so the event needs to be forwarded for matching to B2. Note, that matching at B1 can be omitted, as no subscription created by the clients located in the broker area of B1 can pass the event GeoCheck.

C. Selecting RPs close to the publishers

With this strategy, the RP for an event is the broker closest to the publisher of that event. Thus, the RP is the LB of the publisher. While this means matching only occurs at a single broker, it also implies that all subscriptions must be distributed to all brokers to which a matching event might be published; subscription updates must also be propagated in a similar fashion. Fortunately, the subscription geofence can be used to select these RPs because only broker areas intersecting with the subscription geofence might contain clients that pass the subscription GeoCheck (publisher location inside subscription geofence).

Figure 4 shows an example with one subscriber (S) that is located in the broker area of broker B1 and creates three subscriptions—each has a different subscription geofence (SG):

- SG1 does not intersect with the broker area of broker B2 so the subscription does not need to be forwarded to B2.
- SG2 intersects with the broker areas of B1 and B2 so the subscription needs to be maintained at B1 and be forwarded to B2.
- SG3 only intersects with the broker area of B2 so the subscription needs to be forwarded to B2. Note, that the subscription can be discarded at B1, as none of the clients managed by B1 can publish an event that passes the subscription GeoCheck.

After matching the event, it still needs to be distributed to the LBs of subscribers with matching subscriptions as these brokers are the ones communicating with the subscribers.

D. Discussion

It is straightforward to calculate how many inter-region messages can be saved when using our rendezvous node based approaches compared to a flooding solution for a given event/subscription. In both cases, the given event/subscription must only be distributed to the brokers whose broker areas intersect with the corresponding geofence, rather than to all brokers participating. Consider the following example: if one data-distribution broker instance runs in each of the 21 currently available AWS regions, a published event that is only targeted at clients in Europe would have an event geofence that only intersects with the broker areas of 5 AWS regions and thus reduces the amount of inter-node messages by 16 ($>$75%). In case of more dense deployments or events that are only relevant to an even smaller number of subscribers, benefits could become even higher.

We see the most closely related work in two areas: rendezvous-based pub/sub, e.g., [7]–[9], and geo-distributed, location-based pub/sub, e.g., [2], [10]–[12]. Still, none of these approaches aims to use geo-context information to select RPs close to the publishers or subscribers of an event.

With our current approach, each broker has to know about all other brokers and their broker areas. This is plausible, if the number of brokers is limited to, for example, one very scaleable instance per AWS data center. One interesting aspect about rendezvous-based routing is, however, that matching not always has to occur at the RP when there is a network/hierarchy of nodes. This could potentially be used to further extend our approach to support broker deployments in which not all brokers are connected directly, e.g., because some brokers are also running at the Edge.

IV. Conclusion

In this paper, we proposed to make use of geo-context information attached to published messages and subscriptions to select RPs. By doing so, we can ensure that events and subscriptions are matched close to the publishers or subscribers of an event. This can significantly reduce the amount of data that needs to be distributed between geo-distributed broker instances. In the future, we plan to more thoroughly study the effects of our approach on excess data and quantify effects on system characteristics such as the communication latency between IoT devices. We are currently implementing a broker prototype for use case driven experiments and are also preparing a simulation-based study.

References

[1] P. Bellavista, A. Corradi, and A. Reale, "Quality of service in wide scale publish-subscribe systems," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1591–1616, 2014.

[2] R. Banno, S. Takeuchi, M. Takemoto, T. Kawano, T. Kambayashi, and M. Matsuo, "Designing overlay networks for handling exhaust data in a distributed topic-based pub/sub architecture," *Journal of Information Processing*, vol. 23, no. 2, pp. 105–116, 2015.

[3] R. Baldoni, L. Querzoni, and A. Virgillito, "Distributed event routing in publish/subscribe communication systems: a survey," p. 27, 2005.

[4] Sasu Tarkoma, *Publish/Subscribe Systems - Design and Principles*, ser. Wiley Series in Communications Networking & Distributed Systems. Wiley, 2012.

[5] J. Hasenburg and D. Bermbach, "Towards geo-context aware IoT data distribution," in *4th Workshop on IoT Systems Provisioning & Management for Context-Aware Smart Cities (ISYCC)*. Springer, 2019.

[6] ——, "GeoBroker: Leveraging geo-context for IoT data distribution," *Computer Communications*, 2020.

[7] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *Networked Group Communication*, J. Crowcroft and M. Hofmann, Eds. Springer Berlin Heidelberg, 2001, vol. 2233, pp. 30–43.

[8] P. Pietzuch and J. Bacon, "Hermes: a distributed event-based middleware architecture," in *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*. IEEE, 2002, pp. 611–618.

[9] A. Gupta, O. D. Sahin, D. Agrawal, and A. El Abbadi, "Meghdoot: Content-based publish/subscribe over p2p networks," in *Middleware 2004*, H.-A. Jacobsen, Ed. Springer Berlin Heidelberg, 2004, vol. 3231, pp. 254–273.

[10] G. Cugola and J. Munoz de Cote, "On introducing location awareness in publish-subscribe middleware," in *25th IEEE International Conference on Distributed Computing Systems Workshops*. IEEE, 2005, pp. 377–382.

[11] Y. Teranishi, R. Banno, and T. Akiyama, "Scalable and locality-aware distributed topic-based pub/sub messaging for IoT," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–7.

[12] R. Kawaguchi and M. Bandai, "A distributed MQTT broker system for location-based IoT applications," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2019, pp. 1–4.

Towards Online Reinforcement Learning for Self-adaptive Fog Systems

Alexander Palm, Andreas Metzger, Claas Keller and Jan Löber
paluno - University of Duisburg-Essen
Essen, Germany
Email: firstname.lastname@paluno.uni-due.de

***Abstract*—Fog computing uses geographically distributed fog nodes that can supply nearby end devices with low-latency access to cloud-like compute resources. In order to effectively use fog resources, applications deployed on top of such fog nodes need to self-adapt to changing characteristics and availability of the fog nodes, possibly dynamically balancing resource needs and requirements satisfaction. Due to the uncertainty at design time about the actual fog configuration and characteristics at run time, fully specifying the self-adaptation of fog applications is not feasible. This paper makes a first step towards addressing this challenge by introducing the ENACT Online Learning Enabler and experimentally applying it to a representative, self-adaptive cloud application. The ENACT Online Learning Enabler leverages state-of-the-art reinforcement learning algorithms to enable a self-adaptive application to continuously learn and improve its self-adaptation behavior. We give an insight into the conceptual design of the online learning enabler and its capabilities while applying it to a cloud benchmark platform, serving as an initial validation of the approach to be expanded to fog computing.**

***Index Terms*—Self-Adaptation, Reinforcement Learning, Design Time Uncertainty, Cloud Computing, Fog Computing**

I. Introduction

A *self-adaptive* system can modify its own structure, parameters and behavior at run time based on its perception of the environment, of itself and of its requirements. By adapting itself at run time, the system is able to maintain its quality requirements even if the environment changes dynamically [1], [2]. Self-adaptation thereby can help to effectively use fog resources, by building fog applications being able to adapt to changing characteristics and availability of the fog nodes, possibly dynamically balancing resource needs and requirements satisfaction [3].

To develop a self-adaptive system, software engineers have to create *self-adaptation logic* by specifying when and how the system should adapt itself. As an example, a system engineer may specify event-condition-action rules that define which adaptation action is executed in response to a given environment change. Defining self-adaptation logic requires an intricate understanding of the information system and its environment, and how adaptation impacts on system quality [4]–[6]. Among other concerns, software engineers have to anticipate the potential environment changes the system may encounter at run time in order to define how the system should adapt itself in response to these environment changes. However, anticipating all potential environment changes at design time is in most cases infeasible due to *design time uncertainty* [5], [7]. Especially in the context of fog computing, the challenge of anticipating potential environment changes is amplified; e.g., when compared to cloud computing. Due to the high number and high dynamicity of end devices and fog nodes, fully specifying the self-adaptation of fog applications is not feasible.

One emerging way to address design time uncertainty is to employ *online* reinforcement learning [8]–[17]. Reinforcement learning can learn the effectiveness of adaptation actions through interactions with the system's environment.

In this paper we introduce a novel variant of online reinforcement learning, which leverages state-of-the-art reinforcement learning algorithms to enable a self-adaptive application to continuously learn and improve its self-adaptation behavior. This online learning approach is developed as part of the EU project ENACT [18] and delivered as the so called ENACT Online Learning Enabler. We give an insight into the conceptual design of the ENACT Online Learning Enabler and its capabilities. In particular, we apply it to a cloud benchmark platform, serving as an initial validation of the approach to be expanded to fog computing.

The remainder of this paper is structured as follows: Sec. II introduces the ENACT Online Learning Enabler with necessary background information, its motivation and its underlying concept. Sec. III gives an insight into the corresponding tool, while Sec. IV presents the initial validation using a cloud example and a brief introduction on possible applications in the fog computing domain. Sec. V concludes with an outlook.

II. The Online Learning Enabler

A. Background

1) Online Learning: Online learning comprises machine learning techniques that enable a system to learn during operation time. One common technique thereby is Reinforcement Learning (RL), offering approaches to enable a so-called agent to learn an optimal behaviour policy through direct interaction with its environment [9], [10] without the need for offline training data (as in supervised learning). Applying these RL approaches to self-adaptive software systems (SASS) one can interpret the agent as the adaptation logic of the SASS and the environment as the context in which the SASS operates. As it might be expensive and potentially unreliable to formulate adequate adaptation rules that capture all possible

context situations during design time [19], online learning gives a SASS the opportunity to autonomously learn which adaptation actions to perform in certain situations or context states during runtime. For instance, when considering a cloud platform expected to fulfill certain quality requirements like not exceeding a certain latency threshold when serving user requests, online learning can be used to enable this platform to adapt itself (e.g. through runtime reconfiguration) to meet these kind of requirements even in changing context situations (e.g. different workloads).

2) Reinforcement Learning: A model showing the basic concept of RL can be found in Fig. 1 [20]–[22]. Following this model, in RL a so called *agent* (i.e., system in our case) learns how to perform optimally in an unknown *environment*.

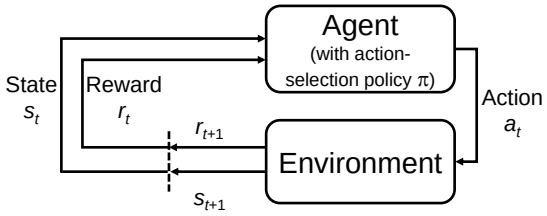


Fig. 1. Conceptual model of reinforcement learning (based on [20])

RL allows solving sequential decision making problems by learning the effectiveness of the agent's actions through interactions with its environment [20], [23]. At each time step t, the agent observes the current state s_t of its environment. Based on this observation the agent selects an action a_t, which may cause the environment to transition into a succeeding state s_{t+1} depending on its dynamics. Furthermore, after performing an action the agent receives a scalar feedback signal in the form of a reward r_t. The overall goal of RL is to learn an optimal *action-selection policy* π by maximizing the agent's cumulative (long-term) rewards. Depending on how rewards are defined for the concrete learning task, the agent's goal may also be to minimize cumulative rewards.

B. Motivation

The ENACT Online Learning Enabler basically aims at two aspects: First, it should ease steps that have to be done during design time of a system. Second, it should enable a system to adapt itself to changing context situations during runtime. Concerning the first aspect: Typically a software engineer needs to foresee all possible context-situations the system might encounter during runtime to formulate adequate rules (e.g. ECA-rules) to enable the system to adapt itself [19]. As this can become very tough especially when the system context changes very often and can vary, the Online Learning Enabler seeks to support a software engineer, so that these traditional steps can be accelerated by providing means to enable the system to directly learn how to behave in certain context situation during runtime. This means that after identifying the continuous environment state variables that influence a context situation, adaptation can be done through parametrization of continuous system variables. To automate this parametrization process at runtime, policy-based Reinforcement Learning is our means of choice, as with this certain RL technique we are able to handle even complex context situations as they might be specified by continuous state and action variables (in contrast to value-based RL approaches like [11]). Furthermore, by directly handling these type of variables an additional step of discretizing the variable spaces and manual tuning of exploration parameters, as it occurs in state-of-the-art approaches, can be avoided.

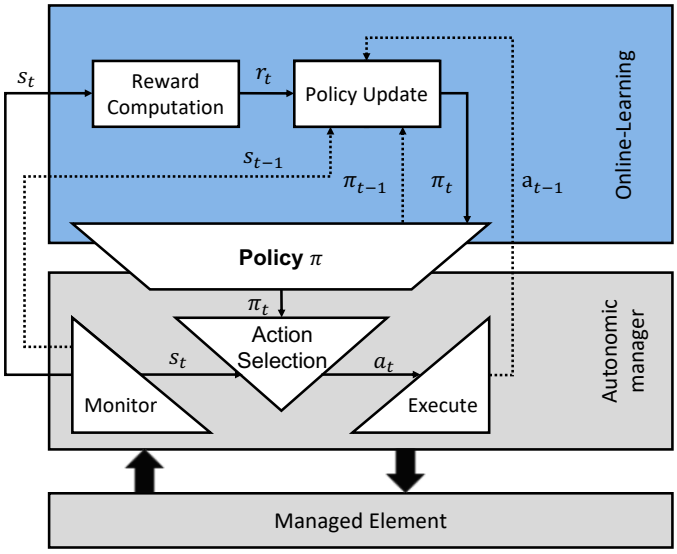


Fig. 2. Conceptual architecture of the Online Learning Enabler

C. Conceptual Architecture

The Online Learning Enabler (OLE) merges the concept of RL with the structure of the MAPE-K model [24]. We still consider the activities *monitor* and *execute* (connecting the adaptation logic with the managed element), but replace *analyze* and *plan* with the policy and the underlying action selection, derived from RL. The resulting conceptual architecture of the OLE is shown in Fig. 2.

The selection of action a_t for the current state s_t is based on the current policy π_t. After action a_t is executed, a new state is reached and replaces the previous state s_t (which is then represented by s_{t-1}). Based on previous experience (comprising past state(s) s_{t-1}, action(s) a_{t-1} and perceived rewards r_t) the current policy is updated after a predefined amount of time steps.

For simplicity, the model shows that the computation of the reward is based on the current state s_t.

III. The Online Learning Tool

Based on the conceptual work, we developed a tool [25], which offers a software engineer the opportunity to enable a system with the ability to adapt itself during runtime. The tool provides specialized components, which are combined by the engineer according to the requirements of the software system. A combination of independent components enables the best possible adaptation to the application and the underlying system. To briefly introduce the field of RL and how to set up

the tool, a documentation component with a running example is provided. A monitoring component provides necessary information about context and system state variables, as well as an opportunity to actually perform actions proposed by the tool. Using the tool an engineer is able to quickly configure a state-of-the-art RL algorithm to solve the identified problem. To do so, the engineer needs to formulate the problem of runtime re-parametrization as a RL problem (i.e., analyze the underlying system to identify states, actions and formulate a suitable reward function) and configure an interface for the communication between the software system and the tool. After starting the learning process he/she can use the monitoring component to supervise the learning process, being able to quickly intervene if something does not go as planned. Technically, the back-end of the tool is implemented in Python using a stable version [26] of the baseline-implementation by OpenAi [27]. The stable-baseline-implementation provides the Proximal Policy Optimization algorithm [28], which is used as a concrete RL technique for online learning. Communication between the underlying system and the tool is established via plain TCP-sockets, using JSON-Objects for data exchange. To enable interactive plotting of the diagrams on the monitoring page, the graphical user interface is set up using the Dash library from Plotly [29], which is based on the JavaScript library React [30]. A snapshot of the graphical user interface is shown in Fig. 3.

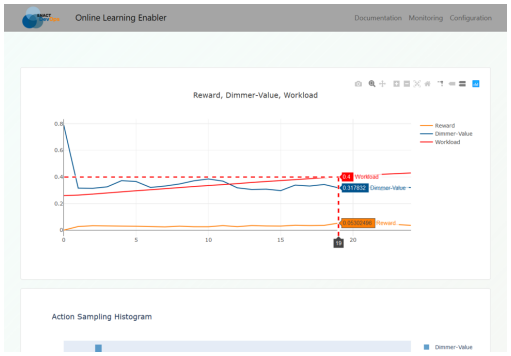


Fig. 3. Snapshot of the GUI of the Online Learning Tool

IV. Application and Validation of Online Learning Tool

A. Initial Validation of Online Learning Tool

To perform an initial validation of our approach we applied the resulting tool to a well-known cloud benchmark platform. We use *Brownout-RUBiS* as a subject system for our validation experiment. Brownout-RUBiS is a self-adaptive variant of a popular web application benchmark, mimicking an auction web application [31]. Users can request information for specific items, which can optionally be served with a list of recommended items based on past auctions provided by the application's recommendation engine. Due to the resource needs of the recommendation engine, Brownout-RUBiS has to balance two quality requirements: maximizing the user experience by providing many recommendations, while minimizing the user-perceived latency. The recommendation engine can be adapted through the ratio of requests being served with recommendations by setting a so-called dimmer value $\delta \in [0,1]$, which represents the per-request probability that the recommendation engine is activated. The dimmer value thus impacts on both quality requirements: A high rate of recommendations increases user experience but at the same time also increases resource needs and thus may increase latency.

The underlying RL-problem is thereby formalized as a Markov Decision Process:

State space: A state s_t comprises the user requests $u_t \in \mathbb{N}^+$, the recommendation ratio $\alpha_t \in [0,1]$ and the latency $\lambda_t \in \mathbb{R}^+$ monitored during a single time step (5s).

Action space: As an action $a_t \in A$ with $A = \delta \in [0,1]$ we defined the adaptation of the dimmer value.

Reward function: The reward function is formulated as $r_t = \alpha_t \cdot f(\lambda_t)$ with $\alpha_t \in [0,1]$ being the recommendation ratio, $\lambda_t \in \mathbb{R}^+$ being the monitored latency and $f(\lambda_t)$ being a utility function defined as follows: $f(\lambda_t) = 1$ if $\lambda_t \leq \lambda_{\max}$; $f(\lambda_t) = 0$ if $\lambda_t > 2 \cdot \lambda_{\max}$; $f(\lambda_t) = -\lambda_t/\lambda_{\max} + 2$ else (= linearly decreasing reward).

By maximizing the reward the Online Learning Tool enables the cloud application to adapt itself to changing workload situations while not violating quality requirements (i.e. latency constraints). Fig. 4 shows the results of an experiment where an On/Off workload pattern has been applied to the cloud platform. It can be seen that the Online Learning Tool is able to adjust the dimmer value in reaction to changes in the workload (which occur every 850 learning cycles). During the first off-phase the dimmer value begins to converge towards a value of 0.55 as the tool learns that this maximizes the reward. After gaining experience on how to properly set the dimmer value during an on-phase, the reward gets maximized during the succeeding phases. Based on the maximization of the cumulative reward the Online Learning Tool enables the cloud platform to serve as many requests as possible with recommendations (i.e. maximizing recommendation ratio) while keeping the average latency of the requests around the predefined threshold of 20ms.

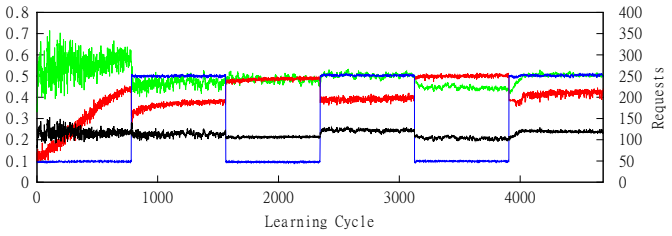


Fig. 4. Application of the Online Learning Tool on a cloud platform with a On/Off workload pattern. State: **blue** = workload, **black** = latency; Action: **green** = dimmer value; Reward: **red**

B. Application in the Fog Computing Domain

Fog computing as an extension of the cloud computing paradigm offers several starting points to use RL techniques

for optimizing decision making at runtime (e.g. optimization of resource allocation, task allocation, etc.) depending on the involved layers (cf. [32]). Typical metrics such as latency, load, etc. are also present in the fog setting and depending on certain constraints they could be used to formulate proper reward functions to drive a certain decision making process. As the context is typically dynamically changing many optimization problems in fog computing bear potential to be formulated as sequential decision making problems.

V. Outlook

The initial validation indicated that the ENACT Online Learning Enabler is indeed able to learn effective self-adaptation actions at run time. However, when expanding the approach from the cloud to the fog setting, one needs to take into account the increased complexity and dynamicity in the fog; e.g., due to a much higher number of collaborating entities. In such a situation, the convergence of the learning process can become a limiting factor. Until reinforcement learning has converged, the system most likely executes inefficient adaptations, because not enough observations have yet been made. To speed up convergence, one avenue to explore is to quickly gather approximate knowledge and then to fine-tune and improve it gradually.

Acknowledgment. Research leading to these results received funding from the EU's Horizon 2020 research and innovation programme under grant agreements no. 780351 (ENACT) and 871525 (FogProtect).

References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *TAAS*, vol. 4, no. 2, 2009.

[2] R. Aschoff and A. Zisman, "QoS-driven proactive adaptation of service composition," in *11th Intl. Conf. on Service-Oriented Computing ICSOC 2011*, ser. LNCS, G. Kappel, Z. Maamar, and H. R. M. Nezhad, Eds., vol. 7084. Springer, 2011, pp. 421–435.

[3] Z. A. Mann, A. Metzger, J. Prade and R. Seidl, "Optimized application deployment in the fog," in *17th Int'l Conference on Service-Oriented Computing (ICSOC 2019), Toulouse, France, October 28-31, 2019*, ser. LNCS. Springer, 2019.

[4] T. Chen and R. Bahsoon, "Self-adaptive and online QoS modeling for cloud-based software services," *IEEE Trans. Software Eng.*, vol. 43, no. 5, pp. 453–475, 2017.

[5] N. D'Ippolito, V. A. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel, "Hope for the best, prepare for the worst: multi-tier control for adaptive systems," in *36th Intl Conf. on Software Engineering, (ICSE 2014)*. ACM, 2014, pp. 688–699.

[6] N. Esfahani, A. Elkhodary, and S. Malek, "A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1467–1493, 2013.

[7] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," in *7th Intl Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)*, 2012, pp. 99–108.

[8] M. D'Angelo *et al.*, "On learning in collective self-adaptive systems: State of practice and a 3d framework," in *14th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2019)*. ACM, 2019.

[9] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, no. 3, pp. 287–299, 2007.

[10] M. Amoui, M. Salehie, S. Mirarab, and L. Tahvildari, "Adaptive action selection in autonomic software using reinforcement learning," in *4th Intl Conf. on Autonomic and Autonomous Systems (ICAS 2008)*. IEEE, 2008, pp. 175–181.

[11] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow," in *7th Intl Conf. on Autonomic and Autonomous Systems (ICAS 2011)*, 2011, pp. 67–74.

[12] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[13] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *17th Intl Symposium on Cluster, Cloud and Grid Computing (CCGRID 2017)*. ACM, 2017, pp. 64–73.

[14] T. Zhao, W. Zhang, H. Zhao, and Z. Jin, "A reinforcement learning-based framework for the generation and evolution of adaptation rules," in *Intl Conf. on Autonomic Computing (ICAC 2017)*. IEEE Computer Society, 2017, pp. 103–112.

[15] A. Moustafa and M. Zhang, "Learning efficient compositions for QoS-aware service provisioning," in *Intl Conf. Conference on Web Services (ICWS 2014)*. IEEE Computer Society, 2014, pp. 185–192.

[16] H. Wang, X. Chen, Q. Wu, Q. Yu, X. Hu, Z. Zheng, and A. Bouguettaya, "Integrating reinforcement learning with multi-agent techniques for adaptive service composition," *TAAS*, vol. 12, no. 2, pp. 8:1–8:42, 2017.

[17] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, 2014.

[18] N. Ferry, A. Solberg, H. Song, S. Lavirotte, J.-Y. Tigli, T. Winter, V. Muntés-Mulero, A. Metzger, E. R. Velasco, and A. C. Aguirre, "Enact: Development, operation, and quality assurance of trustworthy smart iot systems," in *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Springer, 2018, pp. 112–127.

[19] P. Jamshidi, J. Camara, B. Schmerl, C. Kästner, and D. Garlan, "Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots," in *14th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2019)*. ACM, 2019.

[20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[21] F. L. Lewis and D. Liu, *Reinforcement learning and approximate dynamic programming for feedback control*. John Wiley & Sons, 2013, vol. 17.

[22] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[23] T. G. Dietterich, "Machine learning," *ACM Comput. Surv.*, vol. 28, no. 4es, p. 3, 1996.

[24] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[25] A. Palm, "Enact online learning tool," https://gitlab.com/enact/online-learning-enabler, 2020.

[26] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[27] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[29] Plotly Technologies Inc., "Dash library," https://github.com/plotly/dash, 2019.

[30] Facebook Inc., "React library," https://github.com/facebook/react/, 2020.

[31] C. Klein, M. Maggio, K.-E. Arzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *36th Intl Conf. on Software Engineering (ICSE 2014)*. ACM, 2014, pp. 700–711.

[32] J. Bellendorf and Z. Á. Mann, "Classification of optimization problems in fog computing," *Future Generation Computer Systems*, 2020.

Building A Scalable Distributed System For Fog Computing

Vasileios Karagiannis
Distributed Systems Group, TU Wien, Austria
v.karagiannis@infosys.tuwien.ac.at

***Abstract*—The emergence of the Internet of Things along with the trend to create smart applications which automate everyday tasks (e.g., smart homes, smart factories, etc.), have given rise to fog computing. In fog computing, various compute nodes that span from the cloud to the edge of the network, are used for distributing the computations of the smart applications in order to reduce the communication latency, and to improve the bandwidth utilization. Due to these advantages, many fog computing systems have already been proposed in the literature. Most of these systems assume that the participating compute nodes are organized hierarchically, and aim at deploying the applications on these compute nodes, in a way that provides benefits (e.g., low communication latency between the applications and the users). However, the problem of how to organize the participating compute nodes hierarchically, is usually not discussed. This is a serious concern in fog computing because fog computing systems should be able to add new compute nodes dynamically, and still maintain the same hierarchical structure that provides benefits. For this reason, in this paper, we discuss the problem of creating scalable fog computing systems that grow dynamically.**

***Index Terms*—Fog Computing, Edge computing, Distributed Systems, Scalability**

I. Introduction

The advent of the Internet of Things (IoT) has initiated an era with applications that sense the physical world and transfer this information to the cloud for further processing [1], [2]. To facilitate such applications, novel computing paradigms have emerged, two of the most popular being fog and edge computing [3]. Due to the research conducted in the context of fog and edge computing, various models, frameworks, and architectures have been proposed for performing computations at the edge of the network, i.e., closer to the users [4].

Even though fog and edge computing are closely related, there is one characteristic that can be used to separate them. This characteristic is that fog computing–extends–the cloud to the edge of the network. This means that fog computing envisions compute nodes which are organized hierarchically, and span from the edge of the network to the cloud, thereby including the compute resources of the cloud [5]. On the contrary, edge computing focuses on pushing the computations to the edge of the network (e.g, at cloudlets), without necessarily connecting to the cloud compute resources.

Both fog and edge computing systems have been proven useful for providing benefits in use cases such as data stream processing [6], provisioning of services in the IoT [7], smart grids [8], and others [9]. In such systems, the participating compute nodes are organized in specific structures that span over the network, in order to avoid bottlenecks and reduce latency delays [10]. The most widely used structure to implement such systems so far, has been the hierarchical [10]. For example, Fig 1 shows a fog computing system with eight compute nodes which are organized hierarchically in three layers.

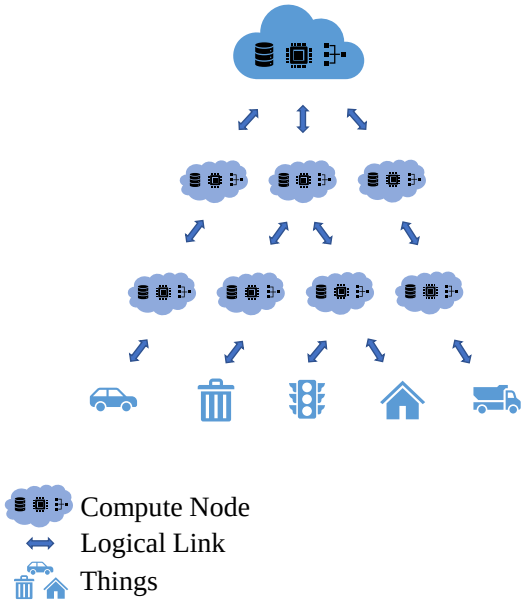


Fig. 1: A hierarchical fog computing system.

In the hierarchical structure, the compute nodes close to the data source are commonly placed low in the hierarchy, while compute nodes that reside farther away, are placed in higher layers. This way, the data is first sent for processing to compute nodes in low layers which reside nearby, and if these compute nodes are unavailable/overloaded, the data moves upwards the hierarchy until there are compute nodes with available compute resources. Many approaches have been designed based on the hierarchical structure, in order to leverage the edge of the network for meeting the application requirements (e.g., regarding bandwidth utilization and communication latency), and for improving the user experience [11].

Despite the popularity and consequently, the growing com-

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation.

munity around fog and edge computing, computing at the edge of the network is still at an early stage, and the research associated with this field is still ongoing. Therefore, several open problems and challenges still exist [12].

Most current approaches for building fog computing systems assume that the participating compute nodes are organized hierarchically in layers, but do not discuss how this is achieved. To evaluate these systems, the logical links among the participating compute nodes are statically configured. This means that most current approaches do not provide mechanisms for adding new nodes dynamically to the hierarchical structure. Even though in small-scale testbeds, static configuration of the logical links among the participating compute nodes is possible, large-scale distributed systems as envisioned in fog computing, cannot be assumed to be statically configured.

Thus, in this paper we discuss the problem of how to organize the compute nodes of a fog computing system in a way that scales dynamically. Moreover, we discuss the various aspects of the system that need to be taken into account while addressing this problem. Such aspects can be, e.g., the proximity among the compute nodes, the resource heterogeneity of the compute nodes, and the fault tolerance of the system in case some of the participating compute nodes fail unexpectedly.

In the following, there is a discussion of related work from the literature in Section II. Afterwards in Section III, we discuss how to address the problem of building scalable fog computing systems along with the various aspects that need to be taken into account, such as: proximity, fault tolerance, and resource heterogeneity. Finally, Section IV concludes this work, and discusses our plans for further research on this topic.

II. Related Work

In this section, we present related work from the literature. Most approaches that aim at building fog computing systems assume that the participating compute nodes are organized hierarchically in layers, e.g., [13]. However, the way whereby this is achieved is usually not discussed. A potential reason that this happens is that such approaches aim at showing the differences between using a centralized compute node, and using distributed compute nodes which are organized hierarchically, i.e., the differences between cloud computing and fog computing.

For instance, Bittencourt et al. [14] discuss resource allocation in a hierarchical structure that consists of three layers. At the bottom of the structure there are the end user devices, in the middle there is a layer of multiple cloudlets, and at the top there is the cloud. This work proposes various scheduling approaches that consider the capacity of the involved resources and the mobility of the users although, the way to build the hierarchical structure is not discussed.

Xia et al. [15] create a system model that includes switches, cloudlets, and cloud compute nodes in a two-layer structure, and tackle the problem of data replication and placement for big data analytics. Notably, for evaluating this approach, the authors build a testbed with multiple virtual machines that communicate with each other. However, the manner that these virtual machines form connections, is not described.

Nguyen et al. [16] propose ICN-Fog which aims at enabling distributed compute nodes to communicate with each other in order to execute applications. According to ICN-Fog, the participating compute nodes form a hierarchical structure. In this structure, the compute nodes from various locations form logical links to the cloud, but also to other compute nodes in proximity. However, the proximity among the compute nodes is assumed without actual measurements. Furthermore, the way that the compute nodes form logical links in order create the hierarchical structure is not discussed. Therefore, this approach can benefit from addressing the proposed problem of enabling the nodes to form logical links dynamically.

Skarlat et al. [7] present FogFrame which is a framework for executing applications in the IoT. To do that, FogFrame creates a hierarchical structure in which there are various compute node at the edge of the network, which are organized in layers. Above these compute nodes, there is a cloud compute node which can be used in case the other compute nodes do not have enough compute resources to execute the applications. In order to organize the participating compute nodes in this structure, FogFrame assumes that all the compute nodes are preconfigured with location coordinates. However, preconfiguring each compute node individually may not be possible in large-scale systems because the number of the participating compute nodes can be too large to allow this. Thus, this approach can benefit from addressing the proposed problem regarding forming the logical links among the compute nodes dynamically.

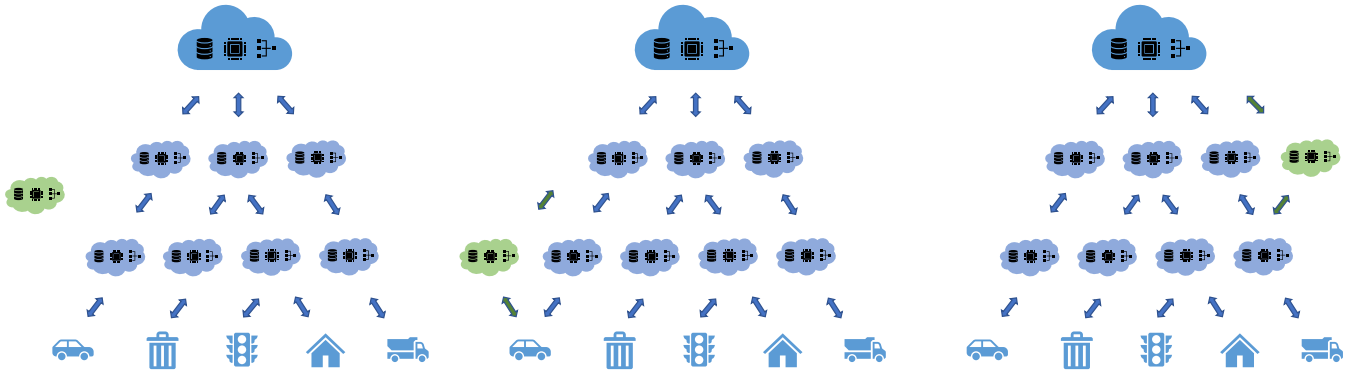
III. Building A Scalable Fog Computing System

Fog computing systems include compute nodes that span from the cloud to the edge of the network. Since this can include a multitude of compute nodes, fog computing systems need to be able to scale to a large degree. Therefore, when a new compute node appears, it needs to be discovered and integrated in the system along with all the preexisting compute nodes.

This can be a complicated task because there may be many different possible places for the new node. For instance, Fig 2a shows a hierarchical fog computing system and a new compute node (with green color) that wants to join in the hierarchy. The problem here is where should the new node be placed.

One possible option is to place the new node in the low layer close to the things, as shown in Fig 2b. Another option is to place the new node in the higher layer closer to the cloud, as shown in Fig 2c. Similarly, there are various alternatives to place the new node in either layer, but with different connections to the nodes of the adjacent layers. When making this decision of where to place each new node that joins a fog computing system, there are various aspects that need to be taken into account. These are described below:

Proximity awareness. A prime objective in fog computing is to reduce the communication latency of latency-sensitive



(a) A new compute node appears (green color).

(b) New compute node is placed in the low layer.

(c) New compute node is placed in the high layer.

Fig. 2: Possible options for a new compute node that joins a fog computing system.

applications [17]. To achieve this, enabling the communication among nodes in close proximity is necessary, because close proximity improves the communication efficiency [18]. Thus, while deciding where each new compute node should be placed, one aspect is to consider which existing nodes are in close proximity, and place the new node in the adjacent layer, enabling their communication and collaboration for processing IoT data.

Resource heterogeneity. Fog computing includes a variety of resource heterogeneous compute nodes which are typically organized hierarchically [5]. In the hierarchy, compute nodes with a large amount of compute resources are commonly placed in higher layers, and aid in the processing of the data from nodes with fewer compute resources, which are placed in lower layers. Therefore, another aspect to consider while deciding where to place a new compute node is the amount of integrated compute resources.

Fault tolerance. Notably, in Fig 2 we can notice that if a node in the middle layer fails or becomes temporarily unavailable, the nodes below may lose connectivity with the rest of the fog computing system. For this reason, fault tolerance mechanisms become essential to fog computing, since node failure might affect the connectivity among the nodes and in turn, the performance of the applications [19]. For this reason, while deciding where to place each new node that joins the system, it is important that the new node makes some additional connections to nodes farther away, which are not used for processing the data, but for maintaining the connectivity in case of failures.

Thus, while deciding where to place each new compute node in a fog computing system, there are various aspects that need to be taken into account. Nevertheless, even though deciding on an appropriate position may not be a simple task, it is evident that this decision needs to be taken dynamically by the system. This is necessary so that a fog computing system can grow dynamically as long as new compute nodes become available.

Therefore, the existing compute nodes, along with the new ones, need to exchange information, e.g., proximity measurements, resource capacities, etc., in order to find appropriate places for the new nodes. Notably, all these messages are the overhead of adding new nodes to a fog computing system. In order to enable fog computing systems to scale to a large degree, this overhead needs to be controlled. For instance, if every new node examines all the possible positions, in large-scale systems, this overhead may grow to a point that it creates bottlenecks, and increases the communication latency among the nodes.

For this reason, in order to create fog computing systems that scale, this overhead needs to remain stable while the number of nodes in the system grows. This ensures that the system will not grow to a point that the overhead creates bottlenecks, since the size of the system does not affect the overhead.

To achieve this, we propose that when a new compute node joins a fog computing system, this new node sends a limited number of messages to existing nodes, and does not explore the whole system to find an appropriate position. This may result in having nodes that are not placed in an optimal position, but it ensures that the overhead will not grow to a point that compromises the scalability of the system. Thus, we note that there can be a trade-off between finding an appropriate position for new nodes, and the resulting overhead.

IV. Conclusion

In this paper, we discuss various aspects of fog computing (such as: proximity awareness, resource heterogeneity, and fault tolerance), and how these aspects need to be taken into account when new compute nodes become available, in order to enable fog computing systems to scale dynamically to a large degree. In our future work, we plan to further analyze the process of adding new nodes to a fog computing system, and to propose concrete algorithms that can be used for building scalable systems for computing that spans from the cloud to the edge of the network.

REFERENCES

[1] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *ICAS Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.

[2] V. Karagiannis, "Building a testbed for the internet of things," pp. 1–92, Alexander Technological Educational Institute of Thessaloniki, 2014.

[3] Y. Liu, J. E. Fieldsend, and G. Min, "A framework of fog computing: Architecture, challenges, and optimization," *IEEE Access*, vol. 5, pp. 25445–25454, 2017.

[4] V. Karagiannis and A. Papageorgiou, "Network-integrated edge computing orchestrator for application placement," in *13th International Conference on Network and Service Management (CNSM)*, pp. 1–5, IEEE, 2017.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Workshop on Mobile Cloud Computing (MCC)*, pp. 13–16, ACM, 2012.

[6] T. Hiessl, V. Karagiannis, C. Hochreiner, S. Schulte, and M. Nardelli, "Optimal placement of stream processing operators in the fog," in *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, IEEE, 2019.

[7] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, and S. Schulte, "A framework for optimization, service placement, and runtime operation in the fog," in *11th International Conference on Utility and Cloud Computing (UCC)*, pp. 164–173, IEEE, 2018.

[8] V. Karagiannis, "Area limitations on smart grid computer networks," *International Journal of Wireless and Microwave Technologies*, vol. 6, no. 3, p. 8, 2016.

[9] V. Karagiannis, A. Venito, R. Coelho, M. Borkowski, and G. Fohler, "Edge computing with peer to peer interactions: Use cases and impact," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, pp. 1–5, ACM, 2019.

[10] V. Karagiannis, "Compute node communication in the fog: Survey and research challenges," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, pp. 1–5, ACM, 2019.

[11] V. De Maio and I. Brandic, "First hop mobile offloading of dag computations," in *18th International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 83–92, IEEE, 2018.

[12] V. Karagiannis, S. Schulte, J. Leitão, and N. Preguiça, "Enabling fog computing using self-organizing compute nodes," in *3nd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, IEEE, 2019.

[13] N. Auluck, O. Rana, S. Nepal, A. Jones, and A. Singh, "Scheduling real time security aware tasks in fog networks," *IEEE Transactions on Services Computing*, 2019.

[14] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[15] Q. Xia, L. Bai, W. Liang, Z. Xu, L. Yao, and L. Wang, "Qos-aware proactive data replication for big data analytics in edge clouds," in *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, pp. 1–10, 2019.

[16] D. Nguyen, Z. Shen, J. Jin, and A. Tagami, "Icn-fog: An information-centric fog-to-fog architecture for data communications," in *2017 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2017.

[17] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.

[18] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[19] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in *1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 115–124, IEEE, 2017.

Fogsy: Towards Holistic Industrial AI Management in Fog and Edge Environments

Patrick Wiener, Philipp Zehnder, Marco Heyden, Patrick Philipp, Dominik Riemer
FZI Research Center for Information Technology
Karlsruhe, Germany
{wiener, zehnder, heyden, philipp, riemer}@fzi.de

***Abstract*—The proliferation of industrial IoT is one of the driving forces that lead to a deluge of generated data bridging the physical and virtual worlds. Consequently, harvesting such data bears a vast potential for companies to realize intelligent, data-driven decisions. In recent years, the decreasing costs for compute resources as well as new decentralized computing paradigms such as fog computing enable companies to invest in artificial intelligence (AI) use cases. However, realizing industrial AI applications is still a major challenge due lack of know-how in AI as well as a complex surrounding infrastructure. In this paper, we present the vision of *Fogsy*, a holistic industrial AI management system aiding to support domain experts to manage analytical AI pipelines from data access, modeling, and training to deployment, monitoring and adaptation in fog and edge environments.**

***Index Terms*—Industrial Artificial Intelligence, Industrial Internet of Things, Fog Computing, Distributed Systems.**

I. Introduction

The steady increase in digitalization in industrial domains such as manufacturing, energy, or logistics has lead to a deluge of generated data with the industrial internet of things (IIoT) as a key enabler bridging the physical and virtual worlds. This offers great opportunities for companies to harvest these new data sources to feed artificial intelligence (AI) based applications to deduce meaningful insights. Typical use cases include both improvements in product and process quality, as well as increased safety in collaborative human-machine scenarios that potentially enable companies to generate competitive cost advantages. For instance, continuous automated visual inspections of produced goods are performed, where a camera scans the product for quality defects to detect deviations in terms of size, shape or component skew. Generally, an industrial AI life cycle begins with data, that are generated by sensors on the edge, served to models for training in the cloud, which are then moved back to the edge for deployment to perform live predictions to autonomously trigger decisions such as stopping a conveyer belt, that again influences sensor measurements. However, realizing industrial AI applications is still a problem, which is especially true for small and medium-sized companies, where they lack (i) dedicated know-how in AI and (ii) the surrounding infrastructure for such end-to-end systems is vast and complex [1]. Particularly in the industrial domain the challenges are manifold. Industrial data are highly heterogeneous in terms of format, protocols, oftentimes inaccurate, and generally provide few incidents for model training, and unlike data of other domains have

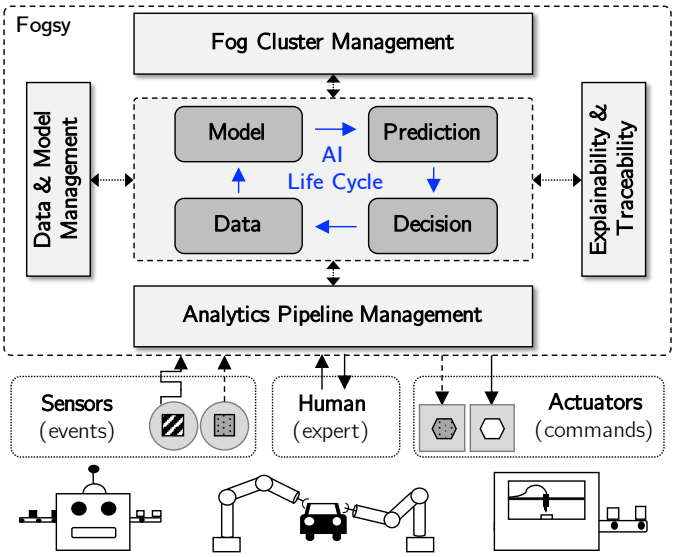


Fig. 1. High-level overview of *Fogsy*.

clear physical meanings resulting in patterns being harder to interpret without domain expertise. Since data is produced on the edge at great volume and speed, e.g., factory shop floor, it induces the challenge of running AI models (optimized in size, prediction speed, throughput) on the edge as well, to overcome the shortcomings of solely centralized cloud deployments in terms of latency, bandwidth or privacy, leading to a vastly distributed infrastructure that needs to be managed. Furthermore, models and their predictions must be explainable for domain experts, especially when the stakes of a single prediction are higher and can be costly.

To address these challenges, we present our vision of *Fogsy*, a holistic industrial AI management system integrating domain experts in the model creation and adaptation process and aiding them to manage analytics pipelines along the cloud-edge continuum as depicted in Figure 1. The contributions of Fogsy are (1) an end-to-end system focused around the AI life cycle from generating training data, to model training and optimization, to deployment, monitoring and model adaptation, (2) a unified fog cluster management to be able to deploy and dynamically adapt pipeline elements of analytics pipelines, e.g., running the right model at the right places, (3) integrating domain experts and their knowledge in an interactive model

creation and refinement process by providing explainability of models and their predictions as well as ensuring overall traceability of the analytical results.

The outline of the paper is as following. In Section II, we show related work in the field of AI management approaches. Section III gives an overview of the conceptual architecture of Fogsy and presents key considerations for the design. Lastly, we summarize and address future work in Section IV.

II. Related Work

To our best knowledge, existing fog computing frameworks do not address the unique challenges of industrial AI management in a holistic manner. Most approaches either only focus on certain aspects of fog computing architectures or present infrastructures which are not specifically tailored for the industrial AI lifecycle and its inherent challenges [2]–[4]. Our previous work has focused on the development of a context-aware, dynamic management infrastructure for stream processing pipelines in the fog [5]. We consider this approach a good foundation and plan on extending it into a holistic solution for industrial AI management based on existing state-of-the-art components and future research. Besides the research community, well-known companies as well as startups offer fog computing services and tools, such as Google Cloud IoT, Amazon Greengrass, Microsoft Azure IoT Edge, IBM Edge Computing, Foghorn[1] or Nebbiolo[2]. So far, however, the ability of domain experts to understand the predictions of the models and improve them through interaction with the system is limited, requires high overhead or is not considered. For these reasons, our vision is to provide a holistic framework around the industrial AI life cycle including AI model management, domain knowledge integration as well as decision explainability and traceability.

III. Fogsy: Holistic Industrial AI Management

In this section, we outline our vision for a holistic industrial AI management framework. Figure 2 illustrates the conceptual architecture consisting of five different layers and their dependencies. The overall goal of the *training layer* is the preparation of AI models suitable for various deployment targets with minimal configuration and adaptation effort, based on techniques such as few-shot learning and multi-objective neural architecture search. The *management layer* handles the management of models and data, orchestrates individual pipeline elements (e.g., pre-processing algorithms and executable models), selects appropriate models for the target execution runtime and handles scheduling, deployment and monitoring. In addition, the management layer derives single hypotheses (e.g., complex process anomalies) by combining multiple, individual models. Models, pre-/post-processing algorithms and adapters to connect data sources are deployed in the *execution layer* on dedicated cloud, fog and edge nodes, each of them managed by a node controller and a node broker to exchange data between pipeline elements running on the same node as well as between nodes, e.g., to send pre-processed data and/or prediction results to a cloud node for storage. Humans (e.g., manufacturing experts) control the system behaviour from the *interaction layer*, which is used to model pipelines, to support training by interactive labeling methods and to explore model explanations as a starting point for continuous model improvement (based on interactive learning). Finally, the *repository layer* stores and provides adapters, pipeline elements, models and training data.

The following subsections describe the scope of some core modules of this architecture (data management, model training and adaptation, explainability, analytics pipeline management and fog cluster management and execution) in more detail.

A. Data Management

The foundation of ensuring high quality AI models is a good data management including data pre-processing and rich meta-information. In industrial settings, however, obtaining useful data is challenging due to high data heterogeneity and the distribution of data sources over multiple locations. First, one has to collect data from machines and sensors, often requiring technical knowledge about the used protocols and formats. In *Fogsy*, we rely on domain experts to connect data sources themselves. As they have the best knowledge about the meaning of data, we can best ensure high quality of data and metainformation. For this purpose, we provide an extensible set of semantic adapters in the repository layer controlled by the data management module. The system instantiates those adapters on edge nodes in the adapter runtime within close proximity to the data source (e.g., corresponding sensors). Furthermore, adapters are capable of reducing the frequency of data streams which is often higher than required for the analytics task. In current solutions data is often stored in a central storage (e.g., in the cloud). However, since compute units can be located near the source (at the edge of the network) and connectivity to the cloud is not always given a centralized storage approach is no longer sufficient. In addition, not all collected data is equally useful for model improvement. For instance, data describing new, unseen situations could have higher relevance than data describing the normal state of the system. All this requires a dynamic management that is capable of routing data in a distributed environment to the processing nodes and deciding which data is relevant for further training, taking the user feedback from the interaction layer into account.

B. Model Training and Adaptation

Model management is essential for efficiently training AI models (e.g., deep neural networks) with high predictive performance, especially when only few training data is available by (i) designing and curating a model repository for industrial tasks as a baseline for ensuring high quality of available models and (ii) building on state-of-the-art technologies for neural architecture search (NAS) [6], [7] and few-shot learning [8], [9]. The model repository structures how models are stored

[1]https://www.foghorn.io (last accessed February 24th 2020)
[2]https://www.nebbiolo.tech (last accessed February 24th 2020)

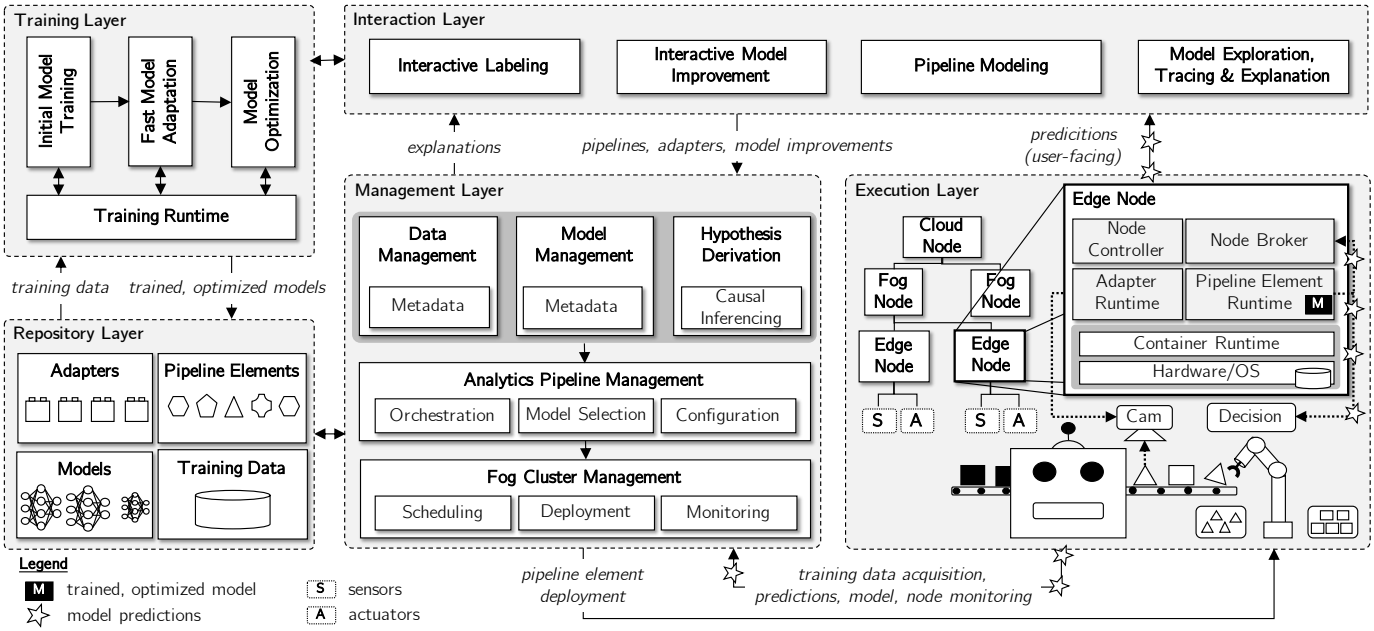


Fig. 2. Conceptual architecture of *Fogsy* showing the various layers shaping the holistic AI management framework.

such that they can be updated when available data distributions change (and model performance or -trust potentially decreases) as well as readily reused and adapted for novel tasks. It thus comprises necessary metadata about the training process of available models, including hyperparameters such as learning rate or batch size, references to the used ground truth and provenance about when model updates occurred. The model repository also needs to provide details about up-to-date evaluations of the model for critical situations to foster trust for end-users. Given a novel task, model management encompasses the complete process from designing the prediction task (e.g., classification, regression or a combination thereof) to fine-tuning the trained models with human-in-the-loop updates. Task design deals with successfully partitioning a given problem, if possible, such that specialized models get be efficiently and quickly trained based on available models in the model repository as well as data in the data repository. It is especially useful when targeted problems are complex, such as quality anomaly prediction for scenes in industrial facilities, where heterogeneous objects need to be classified (e.g. different types of employees, production machines, collaborative robots). Models for specialized tasks are initialized based on available training data and learned model parameters from previous, related tasks available in the model- and data repositories. Few-shot learning (also referred to as meta-learning or transfer learning) deals with efficiently learning such an initialization, resulting in models where few, task-dependent training samples suffice for accurate predictions. As few-shot learning approaches usually assume a fixed model hyperparameters (e.g., the architecture of a neural network), it is essential to combine such algorithms with NAS, where model architectures are adapted and optimized for the novel task to maximize accuracy. Here, multi-objective NAS focusses on searching architectures which, besides accuracy, also account for additional factors, such as computational resources required for training or prediction.

C. Explainability

After specialized AI models have been learned, interactive machine learning [10], [11] enables efficient fine-tuning by incorporating end-users into the training process. By generating post-model explanations [12], [13] of the models (e.g. heatmaps for image processing tasks), the model management system can show the end-user what parts of the input (e.g. an image) the models focussed on in order to enable to reward or penalize the system for different used parts [11]. Finally, model management fosters more expressive explainability by generating contextual hypotheses of available model predictions, such that end-users can understand the reasoning of the system. Contextual hypotheses can be realized as causal graphs [14], enabling to infer correlation and causation among available specialized model predictions. By validating or invalidating individual nodes of the causal graph, the end-user can interact with the available hypothesis in order to change the eventual outcome as well as to train the system.

D. Analytics Pipeline Management

Once models are trained, optimized for edge operation in terms of size, speed, throughput as well as stored in a model repository, they can be leveraged and deployed as part of user-defined analytics pipelines. To this extent, our application model follows a dataflow programming pattern [15], where pipelines are composed of (i) pipeline elements, that are each responsible for a specific task, e.g., data source/sink adapters, numerical filters, or trained models, and (ii) their

interconnections based on the publish-subscribe pattern. The analytics pipeline management receives inputs from the interaction layer, where pipelines are modeled and refined by users, and further orchestrates and configures them. Thereby, this component validates for semantic correctness, selects a model for the chosen task (e.g, classification), and outputs a pipeline description including its requirements. In times, where industrial companies are faced with inherent dynamics and changes in their markets, e.g., shorter product life cycles in the domain of manufacturing, it is inevitable to also reflect and incorporate such modularity, flexibility and adaptability in the analytics workflow.

E. Fog Cluster Management and Execution

Fogsy builts on a unified fog cluster management approach [5] spanning along the cloud-edge continuum by abstracting the necessity of manually managing the heterogeneous and potentially large-scale geo-distributed infrastructure as depicted in the execution layer. At the edge, compute nodes can typically access sensors and actuators as they are deployed within the corporate network to retrieve data or trigger actions as a result of predictions, e.g., sorting out a bad part on the basis of an autonomous visual inspection pipeline. Meanwhile, the cloud functions as a centralized scalable backbone to store analyzed and aggregated data, to (re-)train, adapt, optimize and store AI models for serving and to provide a control plane for system operators to monitor the fog cluster's state, to manage nodes and resources and orchestrate and deploy containerized analytics pipelines. Thus, each infrastructure node is equipped with a node controller, that (i) initially registers node-specific, contextual information (e.g. IP, geolocation, hardware, interfaces to sensors/actuators, etc.) at the central fog cluster management and continuously updates them, (ii) serves as a proxy between deployment decisions and local container runtime, (iii) thus triggering on- and off-loading actions of dedicated pipeline elements. A scheduler then maps the pipeline description and requirements on the underlying infrastructure specifications registered by the individual node controllers, by considering user-defined QoS, contextual information (e.g. priority, privacy), hardware requirements and decides, where to best place individual adapters and pipeline elements, e.g., the optimized AI model for inferencing. As a result, the pipeline execution graph is used for the actual deployment of pipeline elements on the chosen nodes by informing the dedicated node controllers.

IV. Conclusion and Future Work

Industrial AI is receiving great attention due to a manifold of reasons including the accessibility of data, advancements in hardware and the availability of AI software libraries. However, the adoption of AI in the manufacturing sector is still hindered by a number of reasons, which are focused around human involvement, model adaptation and deployment. In this vision paper, we presented *Fogsy*, a holistic industrial AI management system with focus on domain experts ranging from data connection over model training and deployment on the edge, to continuously optimizing models based on user interaction. Future work will evolve around creating a prototypical implementation of the described approaches and evaluate them based on real-world use cases from the industry. Additionally, we plan on integrating the outcome of our work in the open source project Apache StreamPipes (incubating) [3], which was originally initiated by the authors of this paper.

References

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Proc. of the 28th Int. Conf. on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 2503–2511.

[2] D. Santoro, D. Zozin, D. Pizzolli, F. D. Pellegrini, and S. Cretti, "Foggy: A platform for workload orchestration in a fog computing environment," in *2017 IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom)*, Dec 2017, pp. 231–234.

[3] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, and S. Schulte, "A framework for optimization, service placement, and runtime operation in the fog," 10 2018.

[4] A. Brogi, G. Mencagli, D. Neri, J. Soldani, and M. Torquati, "Container-Based Support for Autonomic Data Stream Processing Through the Fog," *Euro-Par 2017: Parallel Processing Workshops*, pp. 17–28, 2018.

[5] P. Wiener, P. Zehnder, and D. Riemer, "Towards Context-Aware and Dynamic Management of Stream Processing Pipelines for Fog Computing," in *2019 IEEE 3rd Int. Conf. on Fog and Edge Computing (ICFEC)*, May 2019, pp. 1–6.

[6] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. of the 35th Int. Conf. on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 4092–4101.

[7] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 2820–2828.

[8] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. of the 34th Int. Conf. on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 1126–1135.

[9] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Advances in Neural Information Processing Systems 32: Annual Conf. on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 2019, pp. 113–124.

[10] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems 30: Annual Conf. on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 4299–4307.

[11] S. Teso and K. Kersting, "Explanatory interactive machine learning," in *Proc. of the 2019 AAAI/ACM Conf. on AI, Ethics, and Society, AIES 2019, Honolulu, HI, USA, January 27-28, 2019*, 2019, pp. 239–245.

[12] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *IEEE Int. Conf. on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 618–626.

[13] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," in *Proc. of the 35th Int. Conf. on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 882–891.

[14] H. E. K. Jr., "Judea pearl, *Causality*, cambridge university press (2000)," *Artif. Intell.*, vol. 169, no. 2, pp. 174–179, 2005.

[15] D. Riemer, L. Stojanovic, and N. Stojanovic, "SEPP: Semantics-based management of fast data streams," in *Proc. - IEEE 7th Int. Conf. on Service-Oriented Computing and Applications, SOCA 2014*. IEEE, Nov. 2014, pp. 113–118.

[3] https://streampipes.apache.org/ (last accessed February 24th 2020)

Living at the Edge: Running IPFS and Ethereum Client Software on Single-board Computers

Simon Krejci, Stefan Schulte
Distributed Systems Group
TU Wien
Vienna, Austria
{s.krejci|s.schulte}@dsg.tuwien.ac.at

***Abstract*—Blockchains and decentralized file systems like the InterPlanetary File System (IPFS) are frequently named as means to share data among different devices and stakeholders in Internet of Things (IoT) scenarios. Very often, the idea is to connect lightweight IoT-based data sources, e.g., sensor nodes, to a ledger or a file system via a Single-board Computer (SBC), i.e., using computational resources at the edge of the network for providing an interface between the data sources and the ledger or file system.**

However, SBCs like a Raspberry Pi naturally possess only limited resources, and therefore may not be suitable to provide an interface between IoT-based data sources and decentralized ledgers and file systems. Within this paper, we evaluate the applicability and performance of common SBCs, namely the Raspberry Pi 3, the Odroid-XU4, and the Intel Galileo Gen2, when used in order to run IPFS and Ethereum client software.

***Index Terms*—Edge computing, blockchain, IPFS, single-board computer**

I. Introduction

The Internet of Things (IoT) is a worldwide network of interconnected devices, which are able to process and store data, and in many cases provide sensor and actuator capabilities [1]. In many IoT scenarios, data distribution and data storage are core functionalities, and different standards and protocols like MQTT are used to distribute data from the sources to the sinks [2]. Recently, the utilization of decentralized file systems like the InterPlanetary File System (IPFS) [3] has been named as a promising means to store data in a distributed way, thus avoiding bottlenecks and relying on a single data storage, e.g., [4], [5]. In many cases, IPFS is combined with blockchain technologies in order to store data in a tamper-proof way, again without the need to rely on a trusted third party, e.g., [6]–[10].

IoT devices like sensor nodes are usually constrained with regard to the available computational power, and/or energy-restricted, since they rely on batteries. Hence, IoT devices are in many cases not powerful enough to directly interact with a blockchain or a decentralized file system [11]. Also, IoT devices may not offer the full software stack necessary to run according client software. In order to use IPFS and blockchains together with IoT devices, a gateway between the devices and the blockchain and IPFS, respectively, is needed. For this, it is very often proposed to make use of cheap yet powerful devices like Single-board Computers (SBCs). Examples for SBCs are the different versions and variants of the Rapsberry Pi, Intel's Galileo and Edison, and Nvidia's different Jetson versions [12].

This work is partially funded by COMET K1, FFG –– Austrian Research Promotion Agency, within the Austrian Center for Digital Production.

However, SBCs again are not necessarily providing a lot of computational resources, so using them as a middle layer between the IoT devices at the edge of the network and the blockchain or IPFS may be too demanding for the SBCs. Hence, as some preparatory work for our research on a fog-IoT middleware which is able to store data items in IPFS and add the according hashes of the data items to an Ethereum-based blockchain, we have conducted some experiments with three SBCs (Raspberry Pi 3, Odroid-XU4, Intel Galileo), showing the resource consumption of standard APIs for IPFS and the Ethereum blockchain.

Within this paper, we present some preliminary results of these experiments. For this, we will give a brief overview of IPFS and blockchains in Section II. Afterwards, we present preliminary results of the experiments in Section III, and conclude the paper in Section IV.

II. Background

The IPFS [3] is a Peer-to-Peer (P2P) distributed file system. It combines concepts of Distributed Hash Tables (DHTs), the filesharing system BitTorrent, the Self-certifying File System (SFS), and the version control system Git. The aim of the IPFS project is to connect all computing devices with one common file system. Furthermore, the nodes in the network do not have to trust each other and no user or node is privileged.

The unique identification of the content stored in the IPFS is done by the *multihash* of the data item. The multihash is a specific format which adds meta-information to the hash of the content. Hence, the multihash comprises a function code which specifies the hash function, the digest length, and the digest bytes.

As mentioned in Section I, the utilization of the IPFS to store (and distribute) IoT data has frequently been named in the related work, very often in combination with blockchain technologies. Generally, a blockchain can be described as a public, distributed ledger, which is built on a P2P network where security is achieved by cryptography. All transactions executed in this network are recorded on this ledger and stored

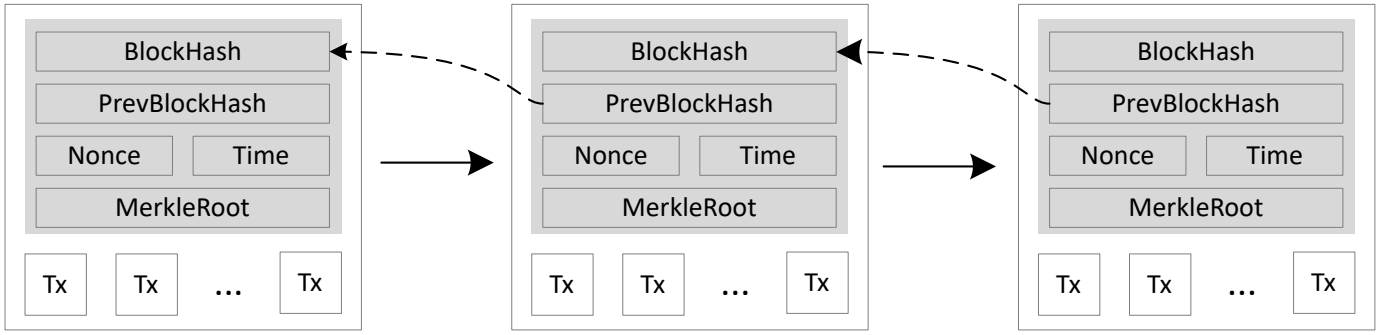


Fig. 1: Simplified Example of a Blockchain (Source: [13])

on every node in the P2P network. Basically[1], the ledger looks as depicted in Figure 1. As it can be seen, several blocks are linked to each other like a chain of blocks. This data structure is the reason for the name blockchain. In the following, we will describe some fundamental aspects of blockchains. For a more detailed description, we refer to [13].

Transactions can be seen as an abstract representation of tokens or other elements in a blockchain. Every transaction has a hash to identify it and a list of inputs and outputs. The input list is used to reference outputs of previous transactions. Notably, an output can only be used as input once in the whole chain. Due to the linking of the transactions, the history of transactions can be traced back.

Besides the transactions, a block also contains a header which comprises several elements. First, the *BlockHash* is the hash value of the block and the *PrevBlockHash* refers to the BlockHash of the previous block. With the PrevBlockHash as a pointer, the data structure of the chain is like a linked list. Therefore, the order of the blocks and transactions can be determined. Furthermore, the blockchain is tamper-proof since the hash depends on the block's content. If some data changes within a block, the hash changes too and thus the references are not correct anymore, which results in an invalid chain. Thus, data manipulations in the data of existing blocks can not be carried without recognition by other peers in a blockchain network.

Because of this feature, blockchains can be used in order to trace monetary transactions in a fully decentralized way – as it is done in cryptocurrencies like Bitcoin [14] or Ethereum [15]. However, blockchains can also be used for other use cases. Especially, second-generation blockchains like Ethereum allow to store almost arbitrary data, however, potentially for a quite high price. This is the reason why blockchain technologies have been frequently named as an important way to share data in IoT settings [16], [17].

As discussed in Section I, IoT devices like sensor nodes are usually not powerful enough to directly participate in a blockchain, e.g., to store data items. This makes it necessary to utilize a gateway like an SBC to run the client software needed to provide interactions between the data sources and the IPFS and the blockchain. Thus, the SBC provides the means of an edge device, which is used to offload computational tasks from the IoT devices. This is a typical use case for SBCs in fog environments [18].

Within the next section, we will evaluate the applicability of SBCs in such settings.

[1]It should be noted that the description here follows the Bitcoin protocol, which is also the foundation for other blockchain protocols, e.g., Ethereum. There are also blockchains which make use of a different basic structure.

III. Experiments

A. Setup

For our experiments, we have chosen three typical SBCs, providing different levels of resources. The SBCs are listed in Table I. To conduct our experiments, we have implemented a software stack for each SBC, as depicted in Figure 2. The software stack contains services to integrate and simulate data items from IoT-based data sources, and to store and distribute these data items using IPFS, a blockchain which is based on the Ethereum protocol, and MQTT. For this, the software stack contains the following components:

- *Sensor Driver*: Provides an interface to a sensor connected to the SBC. This allows to store and distribute data from real-world sensors. In our experiment, the sensor driver has been implemented for the Raspberry Pi, but not the other SBCs.
- *Virtual Driver*: Is used in order to simulate data items which should then be stored and distributed. Using the virtual driver, we can specify the amount of data sources and the amount of sent messages, allowing us to use the virtual drivers in order to execute reproducible performance tests. To be able to test varying loads, we use virtual drivers with 2, 7, 11, and 22 data sources in the

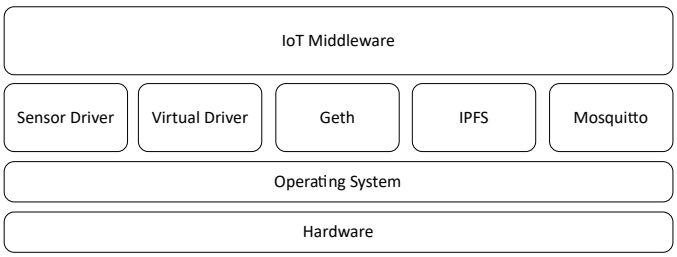


Fig. 2: Setup of an IoT Device.

TABLE I: Selected IoT Devices

	Raspberry Pi 3 Model B	Odroid-XU4	Intel Galileo Gen 2
Chipset	Broadcom BCM2837	Samsung Exynos5422	X1000
CPU	Quad Core @1.2GHz ARM Cortex-A53	Quad Core @2GHz ARM Cortex-A15 Quad Core @1.4GHz ARM Cortex-A7	Single Core @400MHz Intel Quark
Memory	1GB LPDDR2	2GB LPDDR3	256MB DDR3
Ethernet	10/100 MB/s	10/100/1000 MB/s	Available
Storage	MicroSD	MicroSD, eMMC 5.0	MicroSD
OS	Raspbian 8.0	Ubuntu 18.04	Yocto-built Linux

experimental setup. In the current setup, one value per data source is sent every 5 seconds. In total, 60 values are sent, leading to an overall duration of 5 minutes.

- *Geth* implements the Ethereum protocol and serves as entry point into the Ethereum-network, i.e., main, test or private. Geth is run in "light client" mode, i.e., only the headers of the blocks are synchronized and the verification is reduced.
- *IPFS* provides the means to interact with IPFS. For this, we apply the *go-ipfs* client and initialize the client with the "lowpower" profile, which is provided to reduce the overhead.
- *Mosquitto* is installed as MQTT software in order to offer an alternative means for data distribution.

Apart from the listed software, a Python and Java runtime environment is also installed on the respective SBC. Java and Python are needed since our envisioned fog-IoT middleware uses these programming languages.

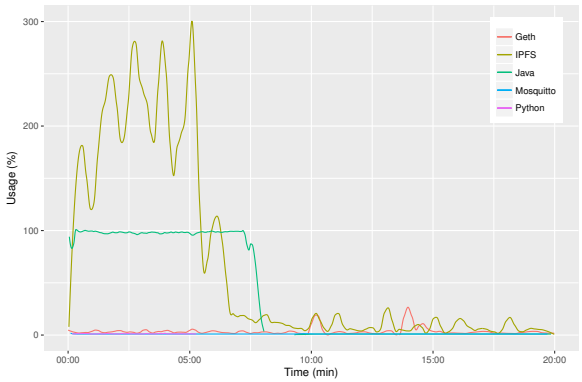
Importantly, the setup already led us to the first important result of our experiments: Since the Intel Galileo does not provide Multimedia Extension (MMX) technology, both Geth and IPFS are not running on this device. Hence, an Intel Galileo is not suitable to used in an IoT scenario where data should be stored and distributed via the Ethereum blockchain or IPFS.

We use the described setup in order to evaluate the *CPU Usage by Process*. To measure the performance, nmon[2] is applied. As it can be seen in Table I, both the Raspberry Pi 3 and the Odroid-XU4 provide quad-core CPUs. Measurements for the CPU utilization are done in nmon with 100% for one core, i.e., with four cores, the CPU utilization may go up to 400%. All experimental runs have been conducted three times. The preliminary results presented in the next subsection therefore provide the average numbers of these runs.

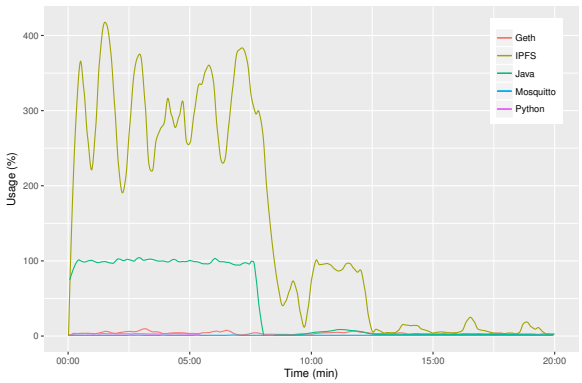
B. Preliminary Results

This section looks at the CPU usage per process whereas the processes Geth, IPFS, Java, Mosquitto and Python are considered. Basically, the measurements for these processes are done every second, however, in the resulting data set the measured timestamps are quite irregular. The reason for this is that the processes are only documented when they use a significant amount of CPU during the specified interval. As written above, nmon calculates the CPU utilization for different cores, i.e., a multi-threaded process may consume more than 100% of one core.

[2]http://nmon.sourceforge.net/pmwiki.php



(a) 2 Data Sources

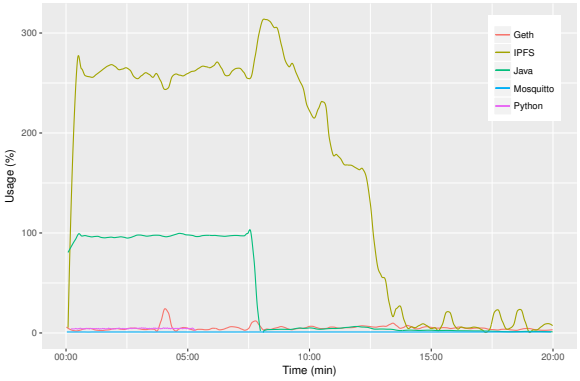


(b) 11 Data Sources

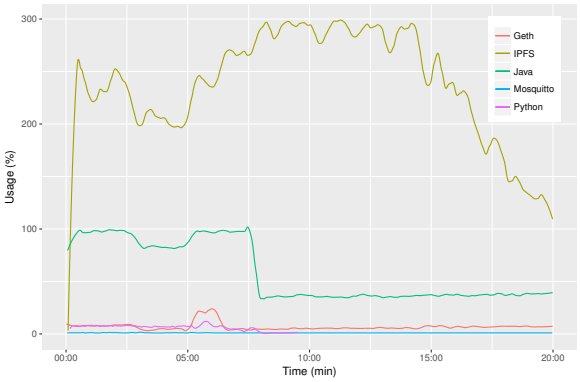
Fig. 3: CPU Usage by Process on Odroid-XU4.

Figure 3 shows the results for the Odroid-XU4 for 2 and 11 data sources, respectively. Notably, despite having more computational resources than the Raspberry Pi 3, the usage of 22 data sources led to the loss of many data items and is therefore not recommended. The first thing to be noticed is the by far highest CPU usage of the IPFS process. The median of the experiment with 11 data sources is 341.77%. If only 2 data sources are needed, the CPU usage drops significantly, namely to 170.71%. The second highest usage is done by the Java process, which executes the fog-IoT middleware and has an approximate median of 99%. All other processes do not have a significant usage.

The same behavior of the processes can be observed on the Raspberry Pi. Figure 4 shows the usage of the processes where



(a) 11 Data Sources



(b) 22 Data Sources

Fig. 4: CPU Usage by Process on Raspberry Pi 3.

IPFS has also by far the highest CPU usage. The maximum usage with 11 data sources is 398.98%[3]. The median in the 22 data sources experiment is 241.21% and in the 11 data sources experiment 262.47%. This can actually be traced back to measurement inaccuracies, i.e., there is no significant difference if 11 or 22 data sources are used.

Due to the results of the performance evaluation, it can be said that IPFS is not a good option to be used on resource-restricted devices like SBCs, while the overhead provided by Geth (running in light client mode) is very small. We have also shown that the number of data items to be added to IPFS increases the resource demand by a very large degree for the Odroid-XU4, but not for the Raspberry Pi 3.

IV. Conclusions

SBCs are frequently used in IoT scenarios in order to connect to data sources, while IPFS and blockchain technologies have gained quite some attention by the research community as means to store and distribute IoT data. However, as has been shown in this paper, using standard client software on SBCs to interact with IPFS is not a good option. Therefore, it is necessary to find more lightweight ways to interact with IPFS.

[3]Notably, the maximum usage is *not* shown in the figures, since the figures represent average numbers over three runs.

Notably, the experiments conducted so far should be seen as preliminary. Apart from the tested devices, there are of course further SBCs, which may be applied in IoT and fog settings and are therefore worth being investigated. This includes, e.g., the different variants and versions of the Raspberry Pi, the Banana Pi, the Nvidia Jetson, and many more. Also, the results may be different if other programming languages (and therefore APIs and client software) are applied.

We have only shown some preliminary results in this paper. We are currently preparing our full experimental outcomes for a more detailed analysis in a follow-up paper.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] A. I. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] J. Benet, "IPFS – Content Addressed, Versioned, P2P File System (DRAFT 3)," *CoRR*, vol. abs/1407.3561, 2014.

[4] S. Muralidharan and H. Ko, "An InterPlanetary File System (IPFS) based IoT framework," in *IEEE International Conference on Consumer Electronics*, pp. 1–2, 2019.

[5] B. Confais, A. Lebre, and B. Parrein, "An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS," in *1st IEEE International Conference on Fog and Edge Computing*, pp. 41–50, 2017.

[6] M. S. Ali, K. Dolui, and F. Antonelli, "IoT Data Privacy via Blockchains and IPFS," in *Seventh International Conference on the Internet of Things*, pp. 14:1–14:7, 2017.

[7] Q. Zheng, Y. Li, P. Chen, and X. Dong, "An Innovative IPFS-Based Storage Model for Blockchain," in *2018 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 704–708, 2018.

[8] Z. Wang, X. Dong, Y. Li, L. Fang, and P. Chen, "IoT Security Model and Performance Evaluation: A Blockchain Approach," in *2018 International Conference on Network Infrastructure and Digital Content*, pp. 260–264, 2018.

[9] N. Rifi, E. Rachkidi, N. Agoulmine, and N. C. Taher, "Towards using blockchain technology for IoT data access protection," in *17th IEEE International Conference on Ubiquitous Wireless Broadband*, pp. 1–5, 2017.

[10] E. Nyaletey, R. M. Parizi, Q. Zhang, and K. R. Choo, "BlockIPFS – Blockchain-Enabled Interplanetary File System for Forensic and Trusted Data Traceability," in *IEEE International Conference on Blockchain*, pp. 18–25, 2019.

[11] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "LSB: A Lightweight Scalable Blockchain for IoT security and anonymity," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 180–197, 2019.

[12] U. Isikdag, "Internet of Things: Single-Board Computers," in *Enhancing Building Information Models: Using IoT Services and Integration Patterns*, pp. 43–53, 2015.

[13] F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[14] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. Whitepaper.

[15] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger." Ethereum Foundation, 2017. Yellow Paper.

[16] K. Christidis and M. Devetikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[17] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Generation Computer Systems*, vol. 88, pp. 173–190, 2018.

[18] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource Provisioning for IoT Services in the Fog," in *9th IEEE International Conference on Service Oriented Computing and Applications*, pp. 32–39, 2016.

Latency in fog computing

Julian Bellendorf
paluno – the Ruhr Institute for Software Technology,
University of Duisburg-Essen
Essen, Germany
julian.bellendorf@paluno.uni-due.de

***Abstract*—Almost every author of scientific publication dealing with the topic of fog computing mentions the low latency when executing computation tasks in the fog nodes as an advantage over executing in the remote cloud. However, this term is often not further defined and aspects which are summarized in it are not clarified. This paper divides the term latency in fog computing into six types with a total of eight sub-types. The aim of this division is to provide an overview of the various aspects of latency and thus to achieve a common understanding of this term.**

***Index Terms*—fog computing, edge computing, latency**

I. Introduction

Fog computing (the term edge computing is used interchangeably by some authors [8]) is a concept in which computing, storage, and network resources are made available distributed at the edge of the network instead of providing resources in a centralized way like in central cloud computing. Those resources are called fog nodes and they are provided in proximity to the end devices. Distributed provisioning of resources close to end devices enables low-latency access to data and services for these end devices [3]. In addition, fog computing enables processing sensor data close to its respective sources so there is no need to send them through the entire network to the cloud for analyzing [8]. However, it is worth mentioning that many authors in the literature have a different understanding of the term latency in the context of fog computing. One reason for this is that some publications focus on fog nodes [5], while others implicitly include into their consideration the concept of cloud computing, which is related to fog computing [6]. Another reason for a different definition of latency is the particular use case that the authors describe.

The contribution of this paper is to provide an overview of various aspects of latency in fog computing. The inclusion or exclusion of these aspects should be clarified for an approach to achieve a common understanding of the term latency in a given context among authors and readers.

A distinction is made between six different types of latency in [2]. Those types are 1)"Time of data transfer between end device and fog node", 2)"Time of executing a task in the fog node", 3)"Time of executing a task in the end device", 4)"Time of executing a task in the cloud", 5)"Time of data transfer between fog node and cloud" and 6)"Time of migration of applications between fog nodes". The subdivision that is shown in the following section restructures and refines the types of latency presented in this paper.

II. Types of latency in fog computing

Fig. 1 provides an overview of different types of latency in fog computing. The architecture which fog computing is based on essentially consists of three layers. The layer shown on the left-hand side includes the cloud, the layer shown on the right-hand side contains the end devices. In between is the layer of fog nodes. These three layers must be taken into account when considering latency in fog computing. For this reason, these three layers are shown at the top of the figure. For each computation that occurs the decision must be made, whether it can be executed in the end device itself or whether it needs to be offloaded to the cloud or the fog nodes. Reasons for offloading from end devices include reduced processing time or decreased energy consumption. Therefore, "Request processing in the cloud" and "Request processing in fog nodes" as well as "task execution in end devices" are types of latency. Besides the time for executing a task on one of the three layers, the time for data transmission between these layers has to be considered, therefore the figure contains the types "Data transmission between cloud and fog nodes" and "Data transmission between fog nodes and end devices". The following subchapters describe in detail the types that are shown in the diagram. Since the layer of the fog node is the focus of this paper, processing within the cloud and within the end devices is not considered in detail.

A. Request processing in fog nodes

Baresi et al. define the three terms acquisition delay, allocation delay, and execution delay, which make up the latency for processing a request send by the end devices to the fog nodes [1]. Acquisition includes downloading and installation of services by the fog nodes. By downloading services on demand, resources can be saved when they are not requested. Allocation means the distribution of downloaded services over a set of fog nodes. Execution delay occurs for a task after acquisition and allocation. Depending on the type of a fog node, more can be added to these 3 types of latency. For example, Zhang et al. describe a fog node consisting of a coordinator and several servers [12]. The coordinator distributes incoming tasks over the servers. Additional latency can occur when data is transferred between the coordinator and the servers.

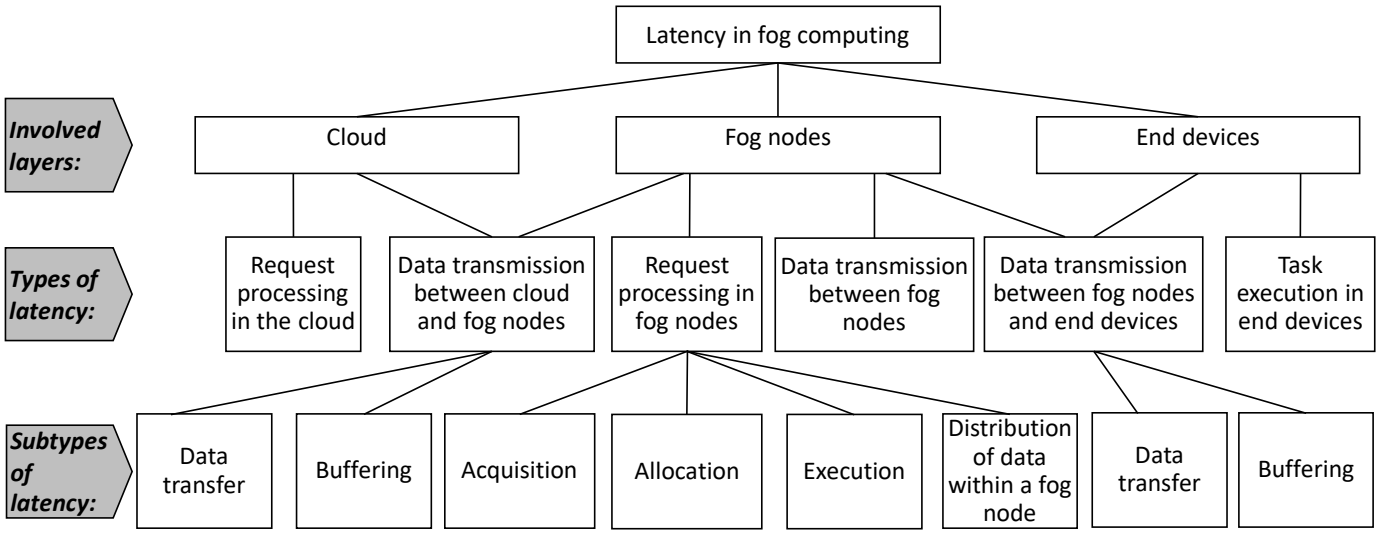


Fig. 1. Overview of different types of latency in fog computing

B. Data transmission between fog nodes

Another type of latency occurs when transmitting data between fog nodes. There can be several reasons for data transfer between fog nodes. One example is the formation of a cluster of fog nodes to bundle their resources and distribute computational tasks over them [7]. The objective of clustering could be to minimize energy consumption during task execution. It is also possible that the resources provided by a fog node are not sufficient to execute a task. In this case, the task could be offloaded to a neighboring fog node [9].

In addition, there is the case that the end device is assigned a virtual machine on the fog node that stores its data and processes its requests. If the end device moves away from this fog node, the associated virtual machine could be migrated to a new fog node to follow the route of the end device. Another type of latency also arises with such a migration [10].

C. Data transmission between fog nodes and end devices and between cloud and fog nodes

In addition to the latency for executing a task, there is latency for the transmission of data between the three layers. On the one hand for the transmission between the end device and the fog node and on the other hand for the transmission between the fog node and the cloud [4]. The latency for data transmission is made up of two parts. The first part is the transfer of the data. This includes the time between the sending of the first symbol by the sender and the reception of the last symbol by the receiver. The second part includes the time that the sent data is in the buffer [11]. The data can be in the buffer of the sender before sending (if the limit of the bandwidth is reached) or in the buffer of the receiver after sending (if the latter is not yet ready for processing).

III. Conclusion and future work

This paper provides an overview of different types of latency that arise in and during transmission between the three layers of fog computing. To achieve a common understanding of the latency in a use case, it can help to go through these types and make clear which ones are relevant in the respective use case. This relevance depends on the one hand on the layers involved and on the other on the type of fog node concerned. For future work a distinction between the different types of energy consumption can be considered, since for energy consumption similar types can be identified as for latency [2].

References

[1] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, pp. 1–21, 2019.

[2] J. Bellendorf and Z. Ádám Mann, "Classification of optimization problems in fog computing," *Future Generation Computer Systems*, 2020.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[4] M.-H. Chen, M. Dong, and B. Liang, "Joint offloading decision and resource allocation for mobile cloud with computing access point," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 3516–3520.

[5] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Transactions on Communications*, vol. 66, no. 4, pp. 1594–1608, 2018.

[6] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2017.

[7] J. Oueis, E. C. Strinati, S. Sardellitti, and S. Barbarossa, "Small cell clustering for efficient distributed fog computing: A multi-user case," in *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*. IEEE, 2015, pp. 1–5.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[9] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, 2017.

[10] H. Yao, C. Bai, D. Zeng, Q. Liang, and Y. Fan, "Migrate or not? exploring virtual machine migration in roadside cloudlet-based vehicular cloud," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 18, pp. 5780–5792, 2015.

[11] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4642–4655, 2018.

[12] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, 2018, pp. 73–80.