Towards a Replication Service for Data-Intensive Fog Applications

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

ABSTRACT

The combination of edge and cloud in the fog computing paradigm enables a new breed of data-intensive applications. These applications, however, have to face a number of fog-specific challenges which developers have to repetitively address for every single application. In this paper, we derive a set of requirements for a replication service that aims to simplify the development of data-intensive fog applications which are caused by the highly distributed and heterogeneous operation environment. Furthermore, we propose the design for such a service which addresses our requirements.

CCS CONCEPTS

• **Information systems** → **Data management systems**; Middleware for databases; • **Computer systems organization** → *Distributed architectures.*

KEYWORDS

Fog Computing, Data Management, Replication Service

ACM Reference Format:
Jonathan Hasenburg, Martin Grambow, David Bermbach. 2020. Towards a Replication Service for Data-Intensive Fog Applications. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3341105.3374060

1 INTRODUCTION

Current state-of-the-art applications are typically deployed on top of cloud services; the cloud alone, however, is often not capable enough for emerging application domains such as autonomous driving, 5G mobile applications, eHealth, or the Internet of Things (IoT) [23]. Depending on the use case, reduced end-user latency, bandwidth limitations between sensors and cloud, or privacy challenges force developers to use fog computing instead: the combination of edge and cloud computing but also with optional small- to medium-sized data centers in the network between cloud and edge offers the best from both worlds [2, 20].

Due to these benefits, we have seen a number of fog applications emerge over the last few years, e.g., [4, 8, 12, 17]; however, one would still expect much more adoption of the fog computing paradigm. In [2], a number of possible reasons such as a lack of edge services or even hardware heterogeneity are discussed. Beyond these, we also see the problem of having to "reinvent the wheel": developers need to start virtually from scratch for every fog application to *get data to where it is needed by a multi-tenant application in a highly distributed and heterogeneous environment.*

In this paper, we propose the design of a data replication service that is specifically tailored for this task. Our goal is to allow application developers to specify data placement and data movement using a declarative programming style while the proposed replication service handles actual data distribution across multiple fog nodes. Therefore, we make the following contributions[1]:

- We identify a set of requirements for a replication service that aims to simplify the development of data-intensive fog applications (section 2).
- We propose and describe the design of a novel replication service that addresses the identified requirements (section 3).

We also discuss related work (section 4) before drawing a conclusion (section 5).

2 REQUIREMENTS

Data-intensive fog applications encounter a number of challenges; most of them are not new but they are significantly more pronounced than in existing cloud-based systems and, thus, require new solutions. The two most obvious challenges are the geo-distribution and heterogeneity of the runtime infrastructure. While cloud-based systems may run in a few geo-distributed data centers on top of more or less identical VM hardware, a fog-based system runs on a variety of machines. These can range from single board computers such as a Raspberry Pi to clusters of cloud VMs and anything in between, geo-distributed over at least hundreds of sites. For data-intensive applications, this means to handle replication and data distribution in such an environment.

Beyond these, resources at or near the edge are limited so that fog-based systems need to deal with much higher degrees of shared resources, not only at the level of infrastructure resources but also in the software stacks on top. Finally, fog-based systems need to interface with a variety of existing systems. These could be embedded cyber-physical systems at the edge, event brokers of all kinds, or stream processing systems and legacy applications in the cloud.

Ideally, a supporting replication service deals with all these aspects to let application developers experience the same simplicity

[1] An extended version of this paper, including a replication service prototype called FBase, is available as [11].

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03.
https://doi.org/10.1145/3341105.3374060

in the fog that they got used to while building cloud applications. We believe that this ideal situation is not achievable, e.g., complete distribution transparency is not feasible in the presence of faults. A well-designed service, however, should provide suitable abstractions to handle the complexities of, e.g., geo-distribution, while not hiding the fact per se. Based on these premises, we identified four main requirements:

Design for Multi-Tenancy: Due to the higher degree of shared resources near the edge, it is not feasible to run several instances of the replication service (or alternative services) in parallel. Instead, such a service should be designed for multi-tenancy out of the box.

Application-Controlled Data Placement: Applications should be able to declaratively specify the placement of data. This means that it should control the placement of data while the underlying service handles data replication, movement, and distribution where and when necessary. This still exposes the fact that different sites exist but takes the hassle out of it.

Hiding Infrastructure Heterogeneity: An application should not have to worry about the number and kind of available machines at a particular site. Instead, these should be exposed through suitable abstractions, e.g., the total amount of available resources, while the service handles load balancing, scheduling, and resource management.

Ability to Interface with Existing Systems: A data-intensive fog application very likely needs to interact with other applications and systems, particularly at the edge but also in the cloud. The data-handling service should provide this functionality as part of the data management tasks as data-intensive applications will interact with other systems to either ingest or expose data. The application should be able to specify such interfaces in the same declarative way that it uses for data placement.

3 REPLICATION SERVICE DESIGN

The main goal of our service design is to provide applications with the means to control data replication and data flow across geo-distributed sites using a declarative programming style. As a typical example, consider the following application: an IoT sensor produces data at the edge (e.g., a temperature sensor), triggers a local actuator (e.g., a smart blind), and buffers data at a nearby edge node. This data is then replicated to an intermediary node, e.g., at the regional level, where it is merged with data from other sensors, aggregated, and then replicated to the cloud. In the cloud, the aggregated historical data from a multitude of sensors is made available to web-based clients.

In this example, it is irrelevant to the application which physical machine handles the data at each site. As such, we developed the concept of nodes to *hide infrastructure heterogeneity*. A node is a set of machines at a specific geographic location. Nodes self-organize and application clients (or other nodes for that matter) can choose to interact with any machine of a given node. We describe this concept in more detail in section 3.2.

In addition, the *application can control data placement* and data flows in a declarative way. For this, we propose the concept of keygroups which group data items that should be handled in the same way; each keygroup has metadata that describes which nodes are handling the keygroup. Nodes can be involved in two roles (simultaneously): first, as replica nodes which persist data locally and serve application requests (e.g., get or put); second, as trigger nodes which passively listen to any updates on the keygroup data and expose these updates via an event-based interface. One purpose of the latter is to *interface with existing systems*; in our example above, this means triggering the local actuator at the edge and triggering the data merge on the intermediary node.

Node details, the geo-location of sites, and other information are available to applications through our cloud-based naming service. We describe the naming service in more detail in section 3.1 and the details of keygroup and data distribution in sections 3.3 and 3.4.

Finally, since keygroups are entirely isolated from each other, our service is inherently *designed for multi-tenancy*.

3.1 Naming and Configuration Management

For tasks such as access control or infrastructure management, it is necessary to assert that machines have unique IDs and that configuration data is stored with strict consistency guarantees. In contrast, application data can often tolerate eventual consistency [22].

For configuration data, our design comprises a component called *naming service* that handles naming (i.e., assignment of unique IDs) and storage of configuration data. IDs are immutable and are tombstoned when they are no longer needed. This means that other machines can safely cache configuration data and also share it with other machines so that the naming service itself is only involved when adding or removing machines. Overall, the naming service acts as the single point of truth for configuration data and naming in case of conflicts.

As the naming service stores no application data, which is likely to be updated frequently, its load is usually low; hence, it is unlikely to become a scaleability bottleneck. Furthermore, as long as the naming service offers the functionality described above, it can be implemented in various ways, e.g., as a centralized service or using a peer-to-peer (P2P) approach.

3.2 Nodes: Managing Collocated Machines

To hide the complexity caused by infrastructure heterogeneity and geo-distribution, we propose an abstraction for collocated machines called *nodes*, as such machines often have the same purpose and are used to scale-out systems. For instance, in an IoT scenario where sensor data is preprocessed at the edge before sending it to a cloud backend, a machine at the edge has to handle data of only a very limited number of sensors while the cloud backend has to handle the data from all edge devices. An obvious solution to this is to have several (preferably stateless) cloud machines behind a load balancer with a shared storage system. Therefore, nodes are groups of collocated machines at the same geographical site that have a shared purpose and use a shared data storage system for persistence. Overall, this means that our design has two levels of infrastructure abstraction: on a node level, machines are organized as a P2P system of nodes (see also figure 1); within a node, machines are organized as P2P with a shared persistence tier.

This allows us to strictly separate data distribution functionality from scaling mechanisms: nodes have a unique name so that messages and data are always addressed to a node instead of to a specific machine. The machine of the target node that ends up

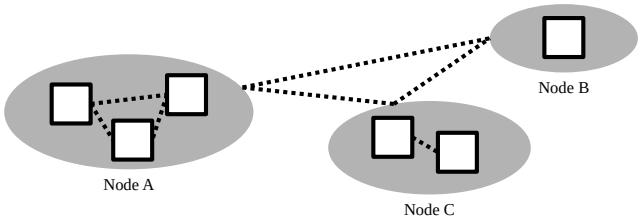

Figure 1: The replication service is a P2P System of Nodes which Comprise a P2P System of Machines Each

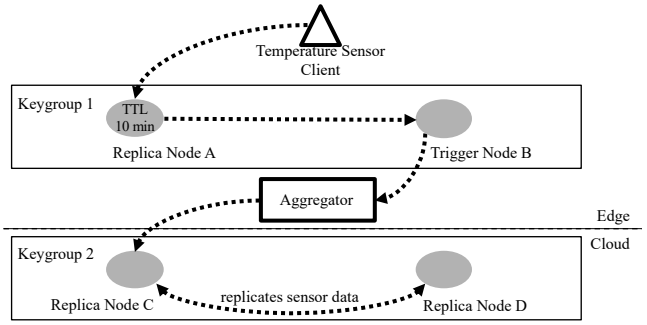

Figure 2: Example: Using Replica and Trigger Nodes to Control Data Distribution

processing this message is hidden from all entities outside of that target node. The strict separation of responsibilities is also helpful for infrastructure membership management: at the node level, there will only be very low churn and even temporary unavailabilities of nodes can be expected to be infrequent due to the built-in redundancy. Also, this helps to keep the load on the naming service at a low level as the more frequent machine churn can be handled within nodes without involving the naming service.

3.3 Keygroups: Encapsulating Logically Coherent Data

Applications often have groups of data items that should be handled in the same way, i.e., they should use the same access policies, should be replicated in the same way, and will often be queried together. Typical examples for this are a sequence of time series values produced by a specific sensor or user records that would be stored in the same table of a relational database. Comparable to the entity groups in Megastore [1], we use the concept of so-called *keygroups* for handling such groups of logically coherent data items which allows natural sharding.

Each keygroup has a globally unique name, some keygroup metadata, as well as the actual data records. The keygroup metadata holds a number of application-defined policies which specify how data should be replicated (see section 3.4) and which clients should have access to the data. This is the key mechanism to give applications control over data distribution.

While keygroup creation and metadata updates (such as giving another party access) need to be confirmed by the naming service, each involved node also stores a copy to reduce load on the naming service and to reduce latency. This naming service dependence ensures that keygroups have globally unique names and that only authorized parties can access the respective data records.

The data records within a keygroup each comprise an unordered set of key-value pairs along with some per-record metadata, in particular update timestamps. As this abstraction is (on purpose) very similar to the BigTable [3] interface, it is relatively simple to utilize any of the widely used, scalable column stores as node persistence tier.

For multi-tenant setups, we propose to use a keygroup naming scheme that maps a keygroup to its tenant, e.g., by including the application id, user id, and data record description. This would lead to names such as "SmartHomeApp.SomeUser.Temperatures".

3.4 Distribution of Data

In our design, there are two primary mechanisms for data distribution that are both specified on a per-keygroup level in the keygroup metadata: replication and transmission. For replication, a set of so-called *replica nodes* is defined that (i) each stores a copy of the respective keygroup's data records and (ii) accepts updates on data records of that keygroup which are then forwarded to all other nodes that are part of the keygroup's node set. Transmission, in contrast, is a mono-directional update propagation mechanism where the so-called *trigger nodes* receive updates from the replica nodes but have only read access to the data records. Trigger nodes specifically exist to integrate legacy applications or external systems such as a stream processing system through an event-based interface that exposes an update stream. As defined in section 2, developers should not have to worry about infrastructure management and heterogeneity. Thus, to replicate data to nodes in a specific geographic area, developers can query the naming service about adequate target nodes as the geo-location is part of the node metadata. Then, the only action left is to add these nodes to the keygroup; the replication service takes care of the rest.

In practice, replica nodes may store incomplete replicas as applications can also specify a time-to-live (TTL) for each replica node of a keygroup, i.e., these replicas will store data records only for a limited period of time. This is particularly useful for edge nodes with limited storage capacity but can also be used to instantiate a buffered data stream that is accessible both as update stream and in an OLTP fashion. The combination of node types and TTL allows applications to specify arbitrary data distribution schemes as needed. See figure 2 for a typical IoT example use case: temperature information is ingested at the edge and buffered there for 10 minutes as defined by Keygroup 1. In addition, an external aggregator component based on a trigger node continuously reads the temperature information from the keygroup and forwards aggregated values to the cloud where they are stored persistently. Keygroup 2 ensures that all the aggregated data is replicated to a second cloud node. Note, that a node can take the role of a replica node and trigger node of the same keygroup at the same time to fulfill both functionalities; in the example, a single node can act as Replica Node A and Trigger Node B.

4 RELATED WORK

Fog computing is still a relatively new computing paradigm. As such, there are many open research questions left, e.g., [2, 7, 19, 20], and there is only a limited number of existing publications in this research area.

Still, a number of papers on data management systems in fog environments exist, e.g., [15, 21, 23]. In difference to our approach, however, these generally do not provide such a fine-grained control of data distribution and replication as offered by our keygroups.

Furthermore, [13, 18] propose data storage systems that make use of edge resources. Both systems, however, make use of a central coordinator that has to handle application data or at least keeps track of changes. Our coordinator, the naming service, only (passively) handles metadata management.

Based on their previous experiments [6], Confais et al. [5] try to make IPFS fog-ready. For this, they modify IPFS so that it does not read from the local file system but rather uses a local (per site) NAS system. While this solves parts of their data locality problems, IPFS is still based on the concept of immutable files which makes it more suitable for content delivery network use cases than for applications with frequent data updates: in such scenarios, IPFS will quickly hit the scalability ceiling.

Finally, a number of papers specifically aims to identify the "optimal" node for data placement. To solve this fog-related problem in geographically distributed systems with many instances, Gupta et al. [9, 10] as well as Mayer et al. [14] propose mechanisms to find suitable replication targets considering the physical data location and node stress levels. Naas et al. [16] formalize this assignment problem and propose a heuristic approach for solving it. Combining such approaches with our keygroups might be a promising avenue to pursue.

A more comprehensive discussion of related work can be found in our extended version [11].

5 CONCLUSION

Emerging application domains such as autonomous driving or the Internet of Things often rely on edge computing, e.g., to circumvent bandwidth limitations or to profit from low latency communication with end users. However, as edge resources are inherently limited, fog computing as a paradigm which combines edge and cloud has recently emerged to combine the benefits of both worlds. While there are already a number of fog applications, we see – among others – the lack of supporting tools and services for running on and integrating both edge and cloud as one of the main reasons for the slow adoption of the fog computing paradigm.

In this paper, we identified a set of requirements that a replication service for data-intensive fog applications should satisfy. Based on these requirements, we proposed and described the design of a replication service that addresses these requirements. This service, for example, allows applications to simply describe how data should be distributed rather than handling data management themselves.

ACKNOWLEDGEMENTS

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 415899119.

REFERENCES

[1] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. 2011. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. In *Proc. of CIDR*.

[2] D. Bermbach, F. Pallas, D. García Pérez, P. Plebani, M. Anderson, R. Kat, and S. Tai. 2017. A Research Perspective on Fog Computing. In *Proc. of ISYCC*. Springer.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: A Distributed Storage System for Structured Data. In *Proc. of OSDI*. USENIX.

[4] Ning Chen, Yu Chen, Yang You, Haibin Ling, Pengpeng Liang, and Roger Zimmermann. 2016. Dynamic Urban Surveillance Video Stream Processing using Fog Computing. In *Proc. of BigMM*. IEEE.

[5] Bastien Confais, Adrien Lebre, and Benoit Parrein. 2017. An Object Store Service for a Fog/Edge Computing Infrastructure Based on IPFS and a Scale-Out NAS. In *Proc. of ICFEC*. IEEE.

[6] Bastien Confais, Adrien Lebre, and Benoît Parrein. 2017. Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure. In *TLDKS*. Springer.

[7] Manuel Díaz, Cristian Martín, and Bartolomé Rubio. 2016. State-of-the-art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing. *Journal of Network and Computer Applications* (2016).

[8] Martin Grambow, Jonathan Hasenburg, and David Bermbach. 2018. Public Video Surveillance: Using the Fog to Increase Privacy. In *Proc. of M4IoT*. ACM.

[9] Harshit Gupta and Umakishore Ramachandran. 2018. Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. ACM, 148–159.

[10] Harshit Gupta, Zhuangdi Xu, and Umakishore Ramachandran. 2018. Datafog: Towards a holistic data management platform for the iot age at the network edge. In *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

[11] Jonathan Hasenburg, Martin Grambow, and David Bermbach. 2019. FBase: A Replication Service for Data-Intensive Fog Applications. In *Technical Report MCC.2019.1*. TU Berlin & ECDF, Mobile Cloud Computing Research Group.

[12] Xueshi Hou, Yong Li, Min Chen, Di Wu, Depeng Jin, and Sheng Chen. 2016. Vehicular Fog Computing: a Viewpoint of Vehicles as the Infrastructures. *IEEE TVT* (2016).

[13] Yi Lin, Bettina Kemme, Marta Patino-Martinez, and Ricardo Jimenez-Peris. 2007. Enhancing Edge Computing with Database Replication. In *Proc. of SRDS*. IEEE.

[14] Ruben Mayer, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. 2017. Fogstore: Toward a distributed data store for fog computing. In *2017 IEEE Fog World Congress (FWC)*. IEEE, 1–6.

[15] Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, and Eyal de Lara. 2017. Cloudpath: a multi-tier cloud computing framework. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 20.

[16] Mohammed Islam Naas, Philippe Raipin Parvedy, Jalil Boukhobza, and Laurent Lemarchand. 2017. iFogStor: an IoT data placement strategy for fog infrastructure. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 97–104.

[17] Subhav Pradhan, Abhishek Dubey, Shweta Khare, Saideep Nannapaneni, Aniruddha Gokhale, Sankaran Mahadevan, Douglas C Schmidt, and Martin Lehofer. 2017. Chariot: Goal-driven Orchestration Middleware for Resilient IoT Systems. *ACM TCPS* (2017).

[18] Mathew Ryden, Kwangsung Oh, Abhishek Chandra, and Jon Weissman. 2014. Nebula: Distributed Edge Cloud for Data Intensive Computing. In *Proc. of IC2E*. IEEE.

[19] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE IoT-J* (2016).

[20] W. Shi and S. Dustdar. 2016. The Promise of Edge Computing. *Computer* (2016).

[21] Jeff Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, and Matt Welsh. 2004. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. In *Techn. Rep. TR-21-04 Harvard University*.

[22] Werner Vogels. 2008. Eventually Consistent. *ACM Queue* (2008).

[23] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward A Lee, and John Kubiatowicz. 2015. The Cloud is Not Enough: Saving IoT from the Cloud.. In *HotStorage*. USENIX.