IoT Data Processing in the Fog: Functions, Streams, or Batch Processing?

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

***Abstract*—When processing IoT data on a large scale, the cloud is no longer sufficient and it has been proposed to move parts of the computation closer to the IoT devices – the so-called fog computing. There are also three basic processing paradigms today that lend themselves to IoT data processing: stream and batch processing as well as serverless functions. Where to place which part of the data processing and which processing paradigm to choose, however, is often unclear.**

In this paper, we give an overview of all three paradigms as well as different data processing use-cases. We use these to derive a decision framework which provides general guidelines for placement of processing and the respectively suitable paradigm when designing a large-scale IoT data processing architecture.

***Index Terms*—IoT, Fog Computing, Edge Computing, Cloud Computing, Data Analytics, Event Processing**

I. Introduction

With the ever increasing number of connected Internet of Things (IoT) devices, traditional data processing approaches in the cloud are no longer feasible in many scenarios. The reason for this lies mainly in bandwidth limitations and latency requirements but additional concerns such as privacy are often also not compatible with cloud-based data processing. Instead, it has been suggested that some data processing should be physically moved closer to these devices in edge and fog computing, or even to the device itself [1]–[3].

However, with a more flexible and distributed infrastructure, it can be hard to decide where and how precisely data should be processed. The vast number of data processing frameworks with varying feature sets and different target use-cases makes such decisions even harder. Therefore, one will typically face two main questions when designing the data processing pipeline for an IoT system: First, there is the decision of where to process which data. For example, data could be filtered on the device, aggregated on the edge, transformed in the fog and ultimately aggregated again in the cloud. Or the edge just relays the data from the device to a specific fog node without any additional transformation. Second, a decision has to be made about the system or service to be used at each step. In our case, the decision between functions of a FaaS platform, stream processing, and batch processing is of particular interest. When deciding on a concrete system, the fact that not all processing tools are available everywhere needs to be considered, e.g., AWS Lambda[1] is only available in the cloud as it is provided as a service.

Our goal with this paper is to help in how to decide which data processing tool or service to use and where to place it in larger Internet of Things systems. Of course, we cannot provide a general architecture that can be applied to every problem, but we will try to provide a framework to make these decisions when designing an IoT data processing pipeline. In our framework, we have divided data processing use-cases into event processing and data analytics. For both types, we present the pros and cons of placing components at specific locations within the edge, fog and cloud and give pointers on when to use stream processors, functions, or batch processing.

In this paper, we will first give a brief introduction to IoT architectures, IoT data, and data processing approaches (section II). Based on this, we will then introduce and explain our decision frameworks for event processing (section III) and data analytics (section IV). Finally, we will discuss related work (section V) and come to a conclusion (section VI).

II. Background

In this section, we will first discuss properties of IoT data, then give an overview of edge and fog computing, and finally describe the three data processing paradigms.

A. Characteristics of IoT Data Processing

IoT data is generated by IoT devices such as sensors, actuators, video cameras, connected cars, and others [4]. And as devices are so diverse, the data they produce is generally not bound to a common structure – a video recording for example is very different from a single measurement by a sensor. However, these devices all have in common that their data is bound to a particular point in time and measured at a specific physical location, whether that location is variable or not. In most cases, temporal and spatial awareness is very important when processing IoT data [1], [5]. Furthermore, IoT data is unbounded in most cases, meaning that the devices never stop sending data. That means that data must be processed as quickly as it is received to prevent an ever-increasing delay in processing [6].

There are also a few features that are not as important in IoT data processing although they may be important in other domains. The first is the persistent storage of every data point. Not only is storing all data an IoT device produces often

[1] aws.amazon.com/lambda

impossible, it is also sometimes not useful. As this data is specific to a point in time, data points quickly become outdated in many scenarios. In these cases, it can be useful to store only an aggregate of the data or only a recent window of data points [7], [8].

Furthermore, there is also the issue of Brewer's CAP theorem. It describes that in a distributed system, which the IoT is, there cannot be consistency, availability and partition-tolerance [9]. In our context, waiving consistency is often the best option and there are two reasons for this. First, availability and partition-tolerance of the components is often more important as the devices should stay working even when disconnected from the network, which often happens with IoT devices (say, for example, a connected car driving through a tunnel) [10]. The second reason is that just like data points can be outdated, most data is only relevant to a small subset of the overall system and not every node needs to have access to a consistent state of the entire system [8], [11]. Given the importance of the temporal characteristics of a data point, however, eventual consistency may be problematic as completeness can also not be guaranteed. Subsequently, it is important to understand that correctness of the overall system cannot be always ensured – still, a large-scale distributed system such as the IoT will simply stop working when forsaking either availability or partition-tolerance.

The data that IoT devices generate can be used in many different ways and how it is eventually processed can differ wildly with different use-cases [12]. However, it can generally be categorized into two groups: *event processing*, where an event triggers a reaction, and *data analytics*, where data is collected and processed to obtain information. Zhang et al. [13] refer to these as "real-time applications with low-latency requirements" and "ambient data collection and analytics" respectively. Of course, these categories overlap as an event can also be interpreted as data and data can trigger certain events [14], [15]. Event processing can have single or multiple sources and is very time-sensitive, meaning that when an event happens, it should be reacted to as fast as possible. This also means that the performed operations are mostly very simple, such as filtering, comparing, or categorizing [15]. Data analytics on the other hand stands out in that more complex operations are performed over data from multiple sources and over a longer period in time, so results are typically not expected instantly [16].

B. Edge, Fog, and Cloud Computing

Traditionally, when processing data from different sources, whether they are located in a company's data center or on-demand SaaS, the ETL pattern, that is Extract, Transform and Load, is often applied, before the stored data can be used in further analysis. Data is first *extracted* from services and other data sources and transferred over the network. In the transformation step, it is cleaned and structured by a common processor before being loaded into a data store for persistent storage. From there, it can be processed further using reporting tools [17].

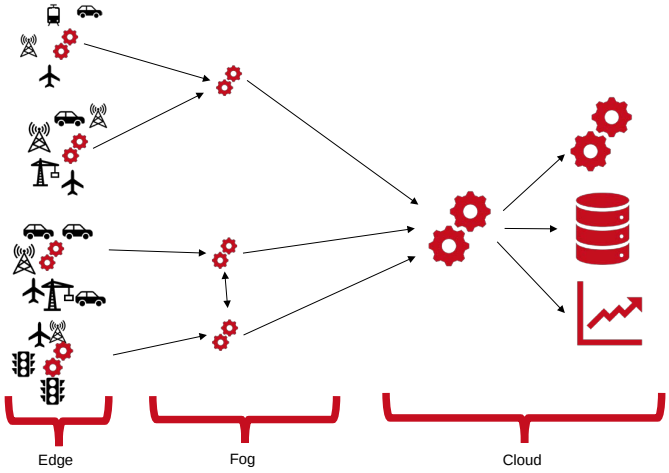

Figure 1. Edge and Fog Computing Topology

In an IoT context, having a single data store and common processor can be helpful for data analytics as it enables a global view of the overall system and is a familiar architecture. However, when millions of IoT devices such as sensors or video cameras constantly send data to a common transformation agent, this agent and the network can quickly become a bottleneck as it can only scale-up to a certain level [18], [19]. Furthermore, this approach is flawed in yet another way. In IoT, latency is everything. When each data point first needs to be sent to the cloud for processing, it takes at least the network round trip time for feedback to get to the actual devices, when in reality, devices may be physically close to each other, e.g., when switching on the light in a smart home scenario.

Fog and edge computing can solve these problems. The idea is to place some data processing components physically closer to the IoT devices. As seen in Figure 1, this creates a multilayered architecture in which processing can be distributed across gateways that are in direct communication with the devices, several fog nodes that bridge the gap to the cloud, and finally the operator in the cloud, that can still transform and load data. Aggregating and preprocessing data closer to the devices reduces the overall network cost and allows for responses with much lower latencies. However, this of course comes at a cost. When moving away from the cloud, its benefits such as cheap and flexible computing power, lower maintenance efforts, or a higher availability are sacrificed [12], [19].

C. Streams, Functions, and Batch Processing

There are three main paradigms in data processing: stream processing, batch processing, and serverless functions as in FaaS (Function-as-a-Service). However, it must be noted that considering systems on a paradigm level generalizes the system design so that actual implementations may differ a lot in their respective designs. Furthermore, it is of course possible, given enough customization, to build a stream processor using a batch processing framework or functions and vice versa. As systems are readily available for each of the paradigms, such a philosophical discussion is beyond the scope of this paper.

a) Functions: As a processing model, the (isolated) function is the simplest one, as it takes a data item as input and computes an output. This assumes that there is no session state and all input data are available before the processing starts. The output is then available at the end, which is very similar to how for example a web server works. While functionality seems limited, a function can be easily implemented with traditional programming techniques and is very scalable. Using a composition of functions, arbitrarily complex processing functionality can be realized. One should, however, try to avoid a data shipping architecture [20]. As stateless components, functions are trivial to scale.

b) Batch Processing: Batch processing engines, such as MapReduce [21] are designed to process bigger datasets efficiently. They use a structured, batched input dataset and perform complex operations on that dataset, for example by distributing computation across multiple nodes. Once the processing is completed, the results, which can be a single value or a new structured dataset, are collected and made available. This approach can be very useful if a lot of related data points need to be processed together, but it also requires all data to be available before the computation can start [22], [23].

c) Streaming: Streaming systems process data as it arrives meaning that results can be obtained nearly instantly. This, of course, comes with some challenges, and only some streaming engines have matured in the last few years, making it a relatively new technology compared to the established batch processing paradigm. One useful feature is that these stream processors have an inherent temporal awareness given that they process data only once at the time it arrives [23]. The fact that data is processed as it arrives is both a strength and a weakness of stream processing. While processed results are available shortly afterwards, the analysis of a particular data point cannot quite consider future values and even for past values there is usually a sliding window defining how far back analysis can go.

III. Event Processing in IoT

In this section, we will discuss characteristics of event processing in more detail. We will also describe what on-device, edge, fog, and cloud computing can provide to support different use-cases and when it makes sense to use either stream processing or functions.

A. Characteristics

As we have described above, the goal of event processing in an IoT context is to react to something that has happened in the physical world, in this case called an event [15], [24], [25]. For example, when an Internet-enabled button has been pressed, this triggers an event. There are a variety of reactions that can be appropriate for an event. Another IoT device could perform an action, a notification could be sent to an external system or service such as IFTTT[2], or the device that originally triggered the event might need to react. In almost all cases, however, the system should be optimized to process an event as fast as possible. In fact, there can often be time constraints for a reaction [15]. Furthermore, events themselves are often very simple at a low level as they often do not carry any data aside from event metadata such as event time, physical location, severity, or others. Of course, however, once a large number of IoT devices all send a single event each at the same time, the total amount of data may still be exceptionally large [26].

[2]ifttt.com

B. Edge, Fog, and Cloud Computing

For event processing, where latency is most important, keeping processing as close to the event source as possible is usually the best strategy as the response latency comprises the round-trip time and the processing time. While the processing time will typically not change dramatically when moving between edge, fog, and cloud since typical reactions are not compute-intensive tasks, the round-trip time is the single influence factor to control overall reaction latency [4], [15]. On the contrary, when moving closer to the device, flexibility and computing cost can become an issue. Generally, cloud computing resources can be scaled up elastically whereas changing compute capabilities on the edge and on-device requires investing money and time [12]. Beyond this, it is important to keep in mind that deciding to move away from the cloud means losing flexibility.

We therefore propose to process events as close to the cloud as possible while still on the shortest path between the event source and the node that needs to react. If the reacting node is a device on the same edge network for example, the edge gateway will be a good choice as long as it provides sufficient resources. On the other hand, if a cloud resource such as an email service needs to be triggered on an event, processing might just as well be done in the cloud: As long as the shortest path between the two nodes is used, the round-trip time will not change. Hence it makes sense to use the most flexible and cheapest computing option as it does not affect event processing time. Furthermore, the closer to the cloud processing happens, the more complex the processing can be. For example, when processing multiple events or event patterns from different devices or edge networks, more computing power is needed. In that case it makes sense to process these events on a node shared by all paths after which the respective paths diverge. In such a case, processing again happens on the shortest path between all emitting nodes while being as close to the cloud as possible, thereby maximizing flexibility and minimizing computational costs.

In the example shown in Figure 2, the event source is an IoT device. Target 1 in this case is the cloud, for example to send a text message using an online service when an event is received. The shortest path to that target is along the edge, fog and cloud. Hence, processing the actual event in the cloud is feasible as it does not increase latency. However, target 2 is a device connected to a different edge gateway but sharing a fog node. As the shortest path to that device does not cross

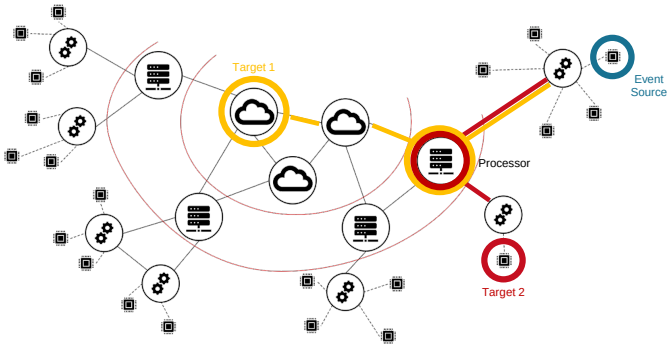

Figure 2. Example: IoT Event Processing

the cloud, sending it along that detour would increase reaction latency. In this case, processing the event on the node marked as Processor will usually be the best solution as it maximizes available compute power without an increase in latency. This is also desirable from the perspective of the edge node next to the event source: Its scarce resources can be used for the things that they are really needed for.

C. Streams, Functions, and Batch Processing

Now that a processing location has been chosen, the next question is how to process the events at that location. We have already identified some characteristics of event processing, such as that it has single or multiple sources, that events tend to be small in size and that their processing is very time-sensitive. We have also discussed that computational cost decreases towards the cloud.

Batch processing is in almost all cases not useful when processing events as it only works on existing data and is not flexible enough to process incoming events in a timely manner. Instead, it is designed to work on larger datasets, which is not a good fit for single small events. Functions seem to be a better fit for event processing. As we have discussed, they are simple and efficient for processing single data items, e.g., a single event. For a given event as the input, they can easily trigger the appropriate reactions as an output. However, once it comes to processing multiple event sources at the same time to detect patterns for instance, the simple processing model can quickly reach its limits as well. As functions are per definition stateless, correlating multiple events that cannot be passed to the function as a single call will only work with an external database for event collection. Stream processing on the other hand is inherently designed for correlation of multiple events. While excessive for simple single-source event processing, when multiple event sources emit numerous events simultaneously, it can be apt to treat this as a stream of events. Of course, stream processing is more computationally intensive, so it should be implemented closer to the cloud. But this should not be an issue as, as we have described, event complexity decreases towards the edge of the network anyway.

We therefore propose to use functions for smaller, single source events, e.g., reacting to someone switching the light on, whether it is in the cloud or on-device, as they are sufficient for that use-case while still being efficient enough to be used even when computing becomes more expensive. When moving away from devices, possibly on edge gateways but mostly in the fog or the cloud, stream processing can be a good option as well, especially once the analysis tasks for the event stream becomes more complex.

IV. Data Analytics in IoT

In this section, we will give a more comprehensive overview of the properties of data analytics, discuss where to run data analytics, and describe how to choose between stream processing, batch processing, and functions.

A. Characteristics

In data analytics, the goal is to extract information from data. In the IoT context, that means using the data the IoT devices produce, which can be sensor measurements, location data, or even a video stream. The main challenge here is the sheer amount of what is produced by these devices and that sending all of that data to the cloud for processing is often not practical, if not impossible, due to bandwidth constraints [3]. On the other hand, in contrast to event processing, processing that data is not as time-sensitive. However, the operations that need to be performed on the data are often more complex, ranging from filtering and aggregation to generating complex reports or predictive analytics. Furthermore, data analytics is usually about correlation of multiple data sources.

B. Edge, Fog, and Cloud Computing

As the required analysis functionality tends to be complex and works on larger data sets, a lot of computational power is needed which is cheaper in the cloud. However, network traffic to the cloud is expensive and/or constrained and should be avoided if possible [27], [28]. Therefore, it makes sense to preprocess – mainly aggregate or compress – the data close to the IoT device before transferring it to the cloud for further analysis. Here, the question is, of course, where to do the preprocessing.

We propose to implement data analytics in the cloud if possible, given the much higher flexibility and lower resource costs. This helps especially when, for example, persistent data storage is required or when new analysis approaches need to be tested. However, of course network constraints can become an issue here. To this end, we also propose that some preprocessing on the data should be done. This can include filtering, transformation, encoding, or aggregation, all of which can save bandwidth. These actions may also be performed dynamically, e.g., a video might be encoded differently based on its contents, which requires analysis of the video data before it is sent to the cloud, so we include dynamic adaption in preprocessing. In each case, preprocessing in terms of data reduction should be run as close to the edge as feasible, i.e., as long as sufficient computer power is available, so as to save as much bandwidth as possible. In the end, it is likely to come down to a question of pricing – compute cost at the edge vs. bandwidth cost [27], [28].

C. Streams, Functions, and Batch Processing

Based on the already identified locations for data analysis and preprocessing, the question is whether to use batch processing, streams, or functions.

As long as preprocessing near the edge or in the fog is limited to filtering, compression, aggregation per data source, or other data reduction mechanisms, functions are a good choice. As soon, however, that preprocessing involves more complex processing tasks that are already part of data analysis – which however will also be not direct at the edge – then stream processing is the most promising paradigm.

For the cloud part where most of the actual data analytics are run, it depends on the concrete requirements. If there are no temporal requirements, e.g., when the analysis results are accessed periodically anyhow, then batch processing might be the best paradigm. This is particularly so since batch processing does not have the windowing restrictions of stream processing and can correlate individual data points with all other data points in a data set and can also run cyclic operator graphs. It supports therefore arbitrary analysis tasks. Stream processing, on the other hand, can provide insights in near real-time which is desirable in many use-cases. As requirements sometimes change, stream processing will typically be the best option whenever the desired analytics task is supported by the processing task.

V. Related Work

As we have discussed already, IoT and, related to that, edge and fog computing, have been a popular research topic in recent times. For example, Bonomi et al. [1] have stressed the importance of fog computing for IoT and Yannuzzi et al. [2] have argued that "fog computing is the natural platform for IoT". Anawar et al. [16] have already presented an extensive overview of the strengths and weaknesses of edge, fog and cloud computing in an IoT context and how it can be used for big data analytics.

Data processing at the edge and in the fog in general have received some attention. For example, Hussain et al. [19] have used data processing in the fog for a real time smart parking system and Kholod et al. [29] have implemented an approach to reduce data transfer from IoT devices to the cloud through local preprocessing.

Meanwhile, Pisani et al. [18] have proposed a framework for bringing code execution from the cloud to IoT devices. Mehdipour et al. [30] have proposed a similar solution for big data analytics.

Similarly, projects such as Apache Edgent [31] and sensorbee [32] have also brought streaming analytics closer to the edge of the IoT network. Furthermore, Sajjad et al [33] have also proposed SpanEdge, a framework that unifies stream processing across the edge, fog and cloud. Yasumoto et al. [34] have also published a survey on real-time processing technologies for IoT. Of course, big data in general has been an important term in Computer Science in recent years. In that context, especially stream processing has gained more traction as a solution for efficiently processing large, constantly evolving data sets. While earlier research, such as that of Kejariwal et al. [35] and Kiran et al. [36], often uses the lambda architecture that combines batch and stream processing, Kreps [37] has proposed using just a streaming engine to do all data processing, which has also been discussed by Lin [38]. Spark Streaming[3] can be seen as a middleground solution between batch processing and streams as it allows users to process so-called microbatches.

But also, event processing and, more specifically, complex event processing, in an IoT context have received more attention. In fact, Govindarajan et al. [15] have motivated the need for moving event processing closer to the edge. Further, Ghosh et al. [24] have even implemented an automated approach to complex event processing in the fog that places the event processor on specific cloud or edge resources to optimize for dataflow latency using a genetic algorithm, thereby obviating the need for a decision to be made when designing the system.

Complementary to our analytic discussion of different paradigms and deployment options, Hasenburg et al. [27], [28] have proposed FogExplorer as a toolkit to interactively explore the QoS and cost effects of different placement options for fog-based functions. This could easily be extended to also cover stream operators but does not discuss the question of choosing a different processing paradigm. Finally, environments such as MockFog [39] can be used to explore different deployment options with various open source stream, batch, and function systems. We propose to use our decision framework to reduce the set of paradigms and deployment options first, then to narrow down the deployment options further with FogExplorer, and finally to validate the decision with a small number of experiments on MockFog.

VI. Conclusion

In this work, we have presented a framework for deciding when and where to implement functions, stream processing, and batch processing in an IoT data processing context. For this purpose, we have discussed the different use-cases of event processing and data analytics, highlighting the capacities of on-device, edge, fog, and cloud computing and demonstrating the strengths and weaknesses of each approach.

Of course, IoT is a wide topic and we are not able to cover all possible data processing use-cases. We have also limited our approach to a hierarchical network of devices, edge gateways, fog nodes, and the cloud, while in reality there may be more or less components, or they may be connected differently. We have also not covered issues of data protection and security, both of which are of great importance in cloud and fog computing and the general IoT. And finally, we have only approached this topic from an analytic perspective, largely without mentioning specific technologies and services available in research and on the market today, with respect to cloud providers, stream processing frameworks, IoT devices, etc. We believe, however, that our framework can be helpful

[3] spark.apache.org/streaming

as general guideline for dealing with IoT data processing and architecting data-intensive fog applications.

REFERENCES

[1] 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*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[2] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Dec. 2014, pp. 325–329.

[3] D. Bermbach, F. Pallas, D. G. Pérez, P. Plebani, M. Anderson, R. Kat, and S. Tai, "A research perspective on fog computing," in *2nd Workshop on IoT Systems Provisioning & Management for Context-Aware Smart Cities (ISYCC)*, vol. 10797. Springer, 2018, pp. 198–210.

[4] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of things patterns," in *Proceedings of the 21st European Conference on Pattern Languages of Programs*. ACM, Jul. 2016, p. 5.

[5] T. Li, Y. Liu, Y. Tian, S. Shen, and W. Mao, "A Storage Solution for Massive IoT Data Based on NoSQL," in *2012 IEEE International Conference on Green Computing and Communications*, Nov. 2012, pp. 50–57.

[6] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor, and Others, "Real time iot stream processing and large-scale data analytics for smart city applications," in *poster session, European Conference on Networks and Communications*, 2014.

[7] N. A. Ali and M. Abu-Elkheir, "Data management for the Internet of Things: Green directions," in *2012 IEEE Globecom Workshops*, Dec. 2012, pp. 386–390.

[8] H. Hromic, D. Le Phuoc, M. Serrano, A. Antonić, I. P. Žarko, C. Hayes, and S. Decker, "Real time analysis of sensor data for the Internet of Things by means of clustering and event processing," in *2015 IEEE International Conference on Communications (ICC)*, Jun. 2015, pp. 685–691.

[9] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002.

[10] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable Fog computing," in *2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*, Jul. 2013, pp. 43–46.

[11] Y. H. Hwang, "IoT Security & Privacy: Threats and Challenges," in *Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security*, ser. IoTPTS '15. New York, NY, USA: ACM, 2015, pp. 1–1.

[12] P. K. D. Pramanik, P. Choudhury, S. Pal, and A. Brahmachari, "Processing IoT Data: From Cloud to Fog-It's Time to Be Down to Earth," in *Applications of Security, Mobile, Analytic and Cloud (SMAC) Technologies for Effective Information Processing and Management*. IGI Global, May 2018, pp. 124–148.

[13] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. A. Lee, and J. Kubiatowicz, "The Cloud is Not Enough: Saving IoT from the Cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015. [Online]. Available: https://www.usenix.org/system/files/conference/hotcloud15/hotcloud15-zhang.pdf

[14] "Events versus Data — The TIBCO Blog," https://www.tibco.com/blog/2010/03/12/events-versus-data/, accessed: 2019-1-31.

[15] N. Govindarajan, Y. Simmhan, N. Jamadagni, and P. Misra, "Event Processing Across Edge and the Cloud for Internet of Things Applications," in *Proceedings of the 20th International Conference on Management of Data*, ser. COMAD '14. Mumbai, India, India: Computer Society of India, 2014, pp. 101–104.

[16] M. R. Anawar, S. Wang, M. Azam Zia, A. K. Jadoon, U. Akram, and S. Raza, "Fog Computing: An Overview of Big IoT Data Analytics," *Proc. Int. Wirel. Commun. Mob. Comput. Conf.*, vol. 2018, May 2018.

[17] D. Fineberg (Intel), "Extract, Transform, and Load Big Data with Apache Hadoop," https://software.intel.com/en-us/articles/extract-transform-and-load-big-data-with-apache-hadoop, Jul. 2013, accessed: 2019-2-11.

[18] F. Pisani, J. R. Brunetta, V. M. do Rosario, and E. Borin, "Beyond the Fog: Bringing Cross-Platform Code Execution to Constrained IoT Devices," in *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct. 2017, pp. 17–24.

[19] F. Hussain and A. Al-Karkhi, "Big Data and Fog Computing," in *Internet of Things*, ser. SpringerBriefs in Electrical and Computer Engineering. Springer, Mar. 2017, pp. 27–44.

[20] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," in *Proceedings of CIDR*, 2019.

[21] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[22] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[23] T. Akidau, "The world beyond batch: Streaming 101," https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101, Aug. 2015, accessed: 2018-12-6.

[24] R. Ghosh and Y. Simmhan, "Distributed Scheduling of Event Analytics across Edge and Cloud," *CoRR*, Aug. 2016.

[25] S. Choochotkaew, H. Yamaguchi, T. Higashino, M. Shibuya, and T. Hasegawa, "EdgeCEP: Fully-Distributed Complex Event Processing on IoT Edges," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Jun. 2017, pp. 121–129.

[26] M. Abu-Elkheir, M. Hayajneh, and N. A. Ali, "Data Management for the Internet of Things: Design Primitives and Solution," *Sensors*, vol. 13, no. 11, pp. 15 582–15 612, Nov. 2013.

[27] J. Hasenburg, S. Werner, and D. Bermbach, "Supporting the evaluation of fog-based IoT applications during the design phase," in *Proceedings of the 5th Workshop on Middlware and Applications for the Internet of Things*. ACM, 2018.

[28] ——, "Fogexplorer," in *Proceedings of the 19th International Middleware Conference (Demos & Posters)*. ACM, 2018, pp. 1–2.

[29] I. Kholod, M. Efimova, A. Rukavitsyn, and S. Andrey, "Time Series Distributed Analysis in IoT with ETL and Data Mining Technologies," in *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer International Publishing, 2017, pp. 97–108.

[30] F. Mehdipour, B. Javadi, and A. Mahanti, "FOG-Engine: Towards Big Data Analytics in the Fog," in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2016, pp. 640–646.

[31] "Apache Edgent," https://edgent.apache.org/, accessed: 2018-12-7.

[32] "SensorBee: An open source stream processing engine for IoT," http://sensorbee.io/, accessed: 2019-1-30.

[33] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "SpanEdge: Towards Unifying Stream Processing over Central and Near-the-Edge Data Centers," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct. 2016, pp. 168–178.

[34] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of Real-time Processing Technologies of IoT Data Streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[35] A. Kejariwal, S. Kulkarni, and K. Ramasamy, "Real Time Analytics: Algorithms and Systems," *CoRR*, Aug. 2017.

[36] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," in *2015 IEEE International Conference on Big Data (Big Data)*, Oct. 2015, pp. 2785–2792.

[37] J. Kreps, "Questioning the Lambda Architecture," https://www.oreilly.com/ideas/questioning-the-lambda-architecture, Jul. 2014, accessed: 2018-12-6.

[38] J. Lin, "The Lambda and the Kappa," *Phi Lambda Kappa Q.*, no. 5, pp. 60–66, 2017.

[39] J. Hasenburg, M. Grambow, E. Grunewald, S. Huk, and D. Bermbach, "MockFog: Emulating Fog Computing Infrastructure in the Cloud," in *Proceedings of the First IEEE International Conference on Fog Computing 2019 (ICFC 2019)*. IEEE, 2019.