AuctionWhisk: Seeing Through the Fog

Example Student
Technische Universität Berlin
example.student@campus.tu-berlin.de

***Abstract*—Why does everything break**

I. Introduction

Function-as-a-service (FaaS), often also referred to as serverless computing, is a relatively new paradigm, where code is only run on demand in isolated environments. This makes it possible to bill users in a fine-grained manner, depending on metrics like number of executions or execution time. [1]

In practice FaaS deployments are often optimized to either run in large datacenters of big public cloud providers [2], [3] (from here on out also referred to as Cloud) or in designated edge-distributed networks[1]. However, as researchers and industries attempt to improve performance and reduce latency further, the attention has also shifted towards alternatives.

One of those potential alternatives is Fog Computing, which we will here define as a fusion of Cloud and Edge Computing where computation resources are deployed on the network paths in between as to split up load over a more diverse infrastructure. That is, each layer between the Cloud and the Edge offers a unique trade-off between resource availability and latency. [4]

Efficiently leveraging the distributed and heterogeneous resources of the aforementioned infrastructure, creates a new set of issues in addition to the preexisting challenges in distributed systems: For instance, determining appropriate scheduling algorithms or optimal allocation of available resources. AuctionWhisk proposed a new way of solving those issues by using an auction inspired approach. Auctions allow for stateless function scheduling without making assumptions on a centralized source of truth by creating a self-contained environment based on developer bids for a function. As in a real auction, the platform can determine the functions that should be scheduled – respective winners of the auction – based on those bids.

In this essay, we make the following contributions:

1) We discuss the advantages and disadvantages of using auctions to determine function placement and execution.
2) We evaluate applications that could potentially benefit or suffer from this approach.

II. Background and Related Work

Although, there has been much research on FaaS and the paradigm has arrived in mainstream technology, with many providers creating their own offering. Be it running in large

[1]For example, Cloudflare's Worker platform is deployed on their globally distributed CDN

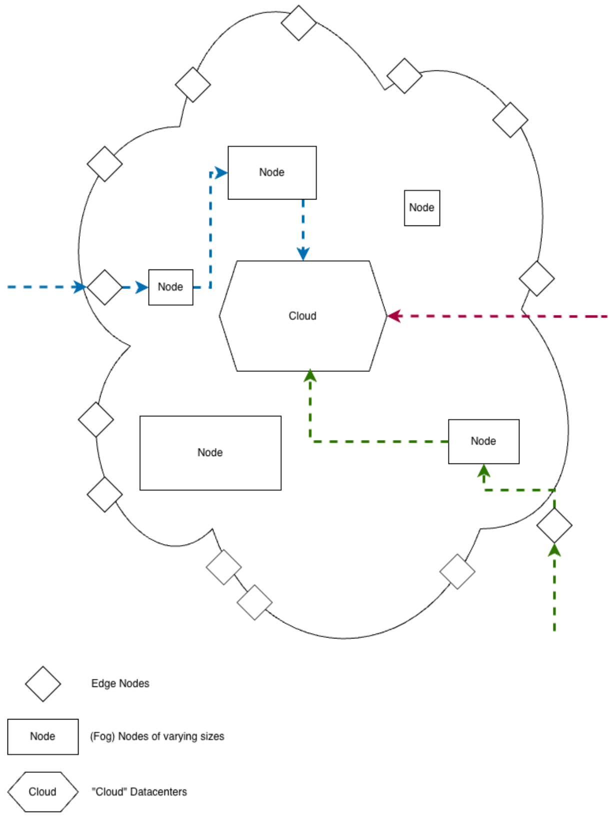

Fig. 1. **Overview of an exemplary Fog system structure**, that shows nodes with varying sizes, where bigger nodes signal higher computational power. They are classified as *Edge Nodes*, *Standard Nodes* and *Cloud*. *Edge Node* are small computational units distributed at the edge (e.g. Raspberry Pis), *Standard Nodes* may be any arbitrary bigger server-site. The *Cloud* is a big datacenter that – in this model – is expected to have virtually infinite resources. Paths in different colors take different routes (to the Cloud) and emphasize the unpredictability of resources when making a request in a fog system.

datacenters (e.g. AWS Lambda, Google Cloud Run Functions, Microsoft Azure Functions) [5] or distributed at the edge (Cloudflare Workers, Fastly Compute) [6]. At the same time, there are only few commercial offerings for FaaS on fog infrastructure [7] and research on the topic is also more scarce.

Deployments in large datacenters and at the edge are conceptually considerably different and offer their respective set of challenges. Nevertheless, one can make assumptions about resource availability in the corresponding environments. To illustrate, a deployment in the Cloud is expected to have virtually infinite resources, which mostly function reliably [2], but a deployment at the edge is expected to have very little, often unreliable resources [8]. However, fog computing resources are often heterogeneous, as shown in Figure 1, with servers closer to the Edge typically possessing lower computational capacity. Consequently, we cannot make the same assumptions about resources here, as we could above. [9]

However, Fog Computing offers three advantages that can be derived from its common setup. Firstly, it promises lower latency by not exclusively running customer code in few, potentially geographically distant datacenters or at the edge where resources are scarce but additionally on infrastructure at the edge and in between. [9] Secondly, it allows for enhanced privacy since user data can be processed relatively local and decentralized. [9] Lastly, by reducing the mean geographical distance between client and server, it minimizes global network bandwidth consumption. [10]

Furthermore, FaaS is traditionally run at a single layer meaning that although requests may be routed through multiple load balancers, proxies, etc., they are only processed at one level of the call graph. Fog computing, on the other hand, commonly uses resources as proxies and servers simultaneously[2]. This is often implemented by checking resource availability at request arrival and forwarding it if resources are insufficient for execution. [10]

III. Auctions in Use

A. *Advantages of auction-based scheduling*

A Distributed System inherently does not have a centralized, reliable source of truth, which creates a set of challenges that researchers have long tried to mitigate. One well-known example is the CAP theorem[3]: It states that a distributed system cannot simultaneously be consistent, available and partition-tolerant [11], which becomes more prevalent in fog environments towards the edge as networking and resources are increasingly unreliable. Auctions alleviate the need for centralized state as they can isolate scheduling decisions on each individual node based on a shared algorithm. This way, expensive synchronization with other nodes or network calls can be avoided, significantly reducing the risk of dependency-induced failures.

Furthermore, employing the dynamic nature of auctions can help to mitigate service downtime in the face of network outages. Networks commonly face partial outages due to a wide array of possible causes. As execution scheduling is often transparent for the client, failures in higher layers of the Fog Stack – e.g. the Cloud – may still affect their function, which is expected to be executed at the edge. In a system that uses auctions to determine which requests will be executed at the edge, the scheduling algorithm becomes more comprehensible for the user, giving them the freedom to decide what to do in such a scenario. On the one hand for example, a failure in the datacenter may prompt a user with high availability requirements to increase their bid, so that the requests are executed at the edge and hence not affected by the Cloud outage. On the other hand, a user with lower availability requirements may be more inclined to tolerate a (partial) outage instead of paying a higher price for guaranteed execution.

Elaborating on this, intelligent bidding strategies can also be applied more broadly to reduce cost while sustaining performance. As execution patterns and load on a public Fog system change over time (often correlated to diurnal or weekly patterns [12]), developers can try to ”beat the market” by adjusting bids in real time. This effect could influence market dynamics, by for example providing a small startup with less financial resources the ability to compete with established co-operations in latency-related issues. To elaborate, a developer could reduce the bids for his globally distributed IoT functions at night – as traffic decreases – in each respective timezone to save money while the functions remain to be executed at the edge.

B. *Disadvantages of auction-based scheduling*

However, the competitive nature of auctions may result in economically unjustified prices for function execution. Although auction-inspired mechanisms can effectively derive a “fair” price for scarce resources based on supply-and-demand dynamics, they introduce additional complexity into the system. As discussed above, such auction markets create opportunities for sophisticated bidding algorithms to optimize the cost–performance trade-off for individual clients. Over time, the use of these algorithms may shift from an optional optimization to a practical necessity, as abstaining from competition can lead to degraded performance or increased costs to maintain equivalent performance. This development, however, contrasts with the serverless promise of the FaaS paradigm, which aims to provide a simple and transparent platform for code execution.

Contrarily, although auctions can decrease latency through statelessness and provide the client with more control over function execution than in traditional fog environments, it can also cause significant performance decrease through auction-induced overhead. To determine a winner in an auction, multiple bids must be considered. In a quickly evolving environment like FaaS, AuctionWhisk achieves this by aggregating an average over all bids in a timeframe and then admitting those function that offered higher bids than the average. This potentially introduces additional latency at two layers. Firstly, as the window is aggregated over a certain time period before the winners are determined, functions that arrive early in a window inherit additional latency by having to wait until the window is closed and a decision is made. Secondly, if a function should be eliminated during an auction, the call is propagated to the next layer on the path where an auction is

[2]Server is a broad term, but here it intends to describe a computing instance that serves requests

[3]Related to this is also the PACELC theorem which differentiates between a partitioned and non-partitioned system

held again. Consequently, the effect can be amplified due to unfortunate timing or an insufficient bid. In a case like this, it is likely that there is not only no advantage gained by using a fog infrastructure, but that performance is worse than it would have been, routing the request directly to the cloud. That is, since on top of network latency, auction latency is introduced as well. [4] This can of course have an impact on the perceived performance-cost ratio of the client.

C. Challenges in FaaS

Some issues that arise in running FaaS on fog infrastructure, cannot be solely attributed to either one of them, but they rather amplify each other. In the following we elaborate on two of them:

Firstly, low resource availability on fog nodes creates tension in combination with the often high resource overhead of function execution. Especially more traditional virtualization technology – such as containerization or simple VMs – have a high memory overhead [13], which causes the scarce resources near the edge to be used less efficiently. In fog environments, functions are commonly small and hence have a considerable overhead in comparison to actual memory footprint. [5]

Secondly, latency increase through time-intensive startups of new virtualization instances is prevalent in fog backends. The process, which is often referred to as a coldstart describes the initialization and creation of a new container, VM or similar virtualized environment, that should execute application code. In correlation with low resources like memory or storage – as discussed above – computation power is commonly poor on many fog nodes. This can prolong resource-intensive coldstarts, thereby also increasing end-to-end latency. This is particular important, when considering the often low-latency requirements that often drives applications into using fog systems.

IV. Auctions

Not only are FaaS and Fog-Computing dynamic concepts because of the continuous research being done in the respective fields but also due to their core paradigms. To deal with the constant change these systems are exposed to, it is important to find solutions that are flexible. To solve the challenges of function placement and load distribution (i.e. where to store a function binary and related data so that it may be executed there) in the heterogeneous, distributed and unreliable environment of Fog Computing, AuctionWhisk employs an auction-inspired approach.

A. Using auctions for decentralized decision-making

Auctions offer an approach for making stateless placement decisions on a particular path from the client to the cloud. Managing a central source of truth in a distributed system requires much effort and many difficult trade-offs and should hence be avoided if possible. Instead, the environment should be able to make decentralized decisions that create a consistent behavior at a high level.

Auctions can act as such mechanisms since they are characterized through their independence of other systems (e.g., in the real world auctions are mostly uninfluenced by the economy and act as a self-contained market). Functions can be efficiently placed with regard to the application's resource needs as well as the cloud providers monetary interests by ranking them by the price a developer is ready to pay. Higher bids are executed at the Edge with lower latency while lower bids are executed in closer proximity to the Cloud where resources are more abundant but latency higher. This model works particularly well in a fog environment as only little advantage is generated by having shared state among the nodes. That is, since the prices should be bound to a location so that individual resource limits are considered. For example, the system does not gain an advantage by comparing bids from developers of latency-sensitive applications in (A) a remote resource-constrained region and (B) a metropolitan area with abundant resources, as the scenarios are hardly comparable.

B. Latency effects of auctions

Although auctions can save costs for the user while maintaining performance, it can also cause significant performance decrease through auction-induced overhead. To determine a winner in an auction, multiple bids must be considered. In a quickly evolving environment like FaaS, AuctionWhisk achieves this by aggregating an average over all bids in a time-frame and then admitting those function that offered higher bids than the average. This potentially introduces additional latency at two layers. Firstly, as the window is aggregated over a certain time period before the winners are determined, functions that arrive early in a window inherit additional latency by having to wait until the window is closed and a decision is made. Secondly, if a function should be eliminated during an auction, the call is propagated to the next layer on the path where an auction is held again. Consequently, the effect can be amplified due to unfortunate timing or an insufficient bid. In a case like this, it is likely that there is not only no advantage gained by using a fog infrastructure, but that performance is worse than it would have been, routing the request directly to the cloud. That is, since on top of network latency, auction latency is introduced as well. [6] This can of course have an impact on the perceived performance-cost ratio of the client.

V. Use-cases

The trade-offs AuctionWhisk makes, enable it to work well in some use cases, whilst potentially performing poorly in others. The following sections summarize the systems advantages and disadvantages by introducing two examples: On the one hand a scenario AuctionWhisk is optimally suited to handle, and on the other hand a situation that that does not

[4] Auction latency also includes serialization, deserialization, IO-operations etc.

[5] Though newer virtualization technologies show improved overhead, they are not broadly established yet

[6] Auction latency also includes serialization, deserialization, IO-operations etc.

perform well. To consider only comparable alternatives and draw a relevant conclusion, we will only consider alternatives that are also based on the FaaS paradigm.

A. A

In the first instance, we will observe an environment where latency is paramount and other aspects, such as cost

B.

stable latency / performance

VI. Conclusion

AuctionWhisk adds a new dimension to FaaS by introducing auctions. Although this opens the paradigm to new opportunities such as cost-optimization through bidding strategies or low-latency through

References

[1] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-52. New York, NY, USA: Association for Computing Machinery, 2019, p. 1063–1075. [Online]. Available: https://doi.org/10.1145/3352460.3358296

[2] M. Brooker, M. Danilov, C. Greenwood, and P. Piwonka, "On-demand container loading in AWS lambda," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 315–328. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/brooker

[3] K. Djemame, M. Parker, and D. Datsev, "Open-source serverless architectures: an evaluation of apache openwhisk," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 329–335.

[4] O. Ascigil, A. G. Tasiopoulos, T. K. Phan, V. Sourlas, I. Psaras, and G. Pavlou, "Resource provisioning and allocation in function-as-a-service edge-clouds," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 2410–2424, 2022.

[5] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 162–169.

[6] N. Ekwe-Ekwe and L. Amos, "The state of faas: An analysis of public functions-as-a-service providers," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, 2024, pp. 430–438.

[7] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada, "Fog function: Serverless fog computing for data intensive iot services," in *2019 IEEE International Conference on Services Computing (SCC)*, 2019, pp. 28–35.

[8] M. Ciavotta, D. Motterlini, M. Savi, and A. Tundo, "Dfaas: Decentralized function-as-a-service for federated edge computing," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, 2021, pp. 1–4.

[9] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, p. 27–32, Oct. 2014. [Online]. Available: https://doi.org/10.1145/2677046.2677052

[10] D. Bermbach, J. Bader, J. Hasenburg, T. Pfandzelter, and L. Thamsen, "Auctionwhisk: Using an auction-inspired approach for function placement in serverless fog platforms," *Software: Practice and Experience*, vol. 52, no. 5, pp. 1143–1169, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3058

[11] M. Kleppmann, "Designing data-intensive applications," 2019.

[12] T. Schirmer, N. Japke, S. Greten, T. Pfandzelter, and D. Bermbach, "The night shift: Understanding performance variability of cloud serverless platforms," in *Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies*, ser. SESAME '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 27–33. [Online]. Available: https://doi.org/10.1145/3592533.3592808

[13] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 419–434. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/agache