AuctionWhisk: Seeing Through the Fog

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

***Abstract*—AuctionWhisk introduces stateless Function-as-a-Service scheduling into Fog Computing by employing auctions, where users can improve application performance by bidding on better resources. Not only do auctions have the potential to improve performance, but they also provide the Function developer with more flexibility and control over the cost-performance ratio. However, the statelessness, as well as auctions, introduce their own trade-offs, that can increase the complexity of using the system and even decrease performance under certain conditions.**

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 in Section III and Section IV
2) In Section V, we then 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 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]

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

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

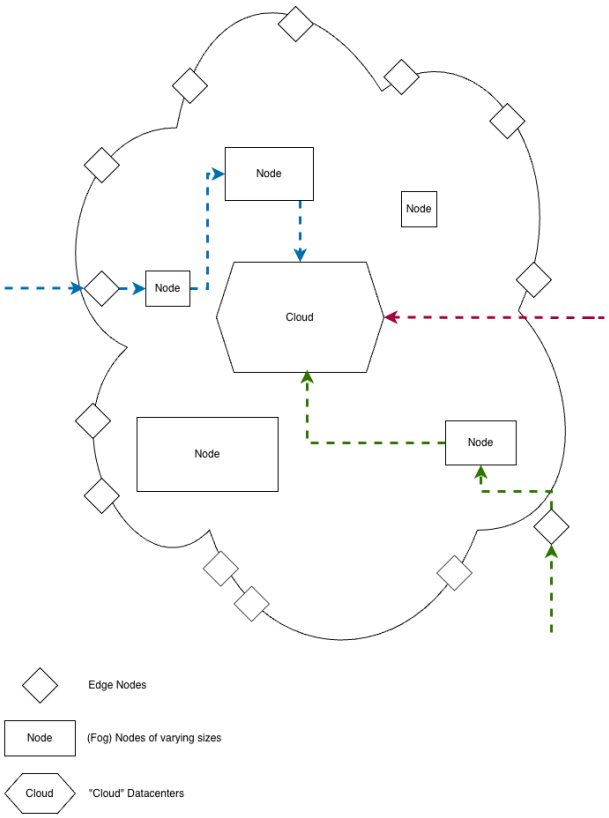

Fig. 1. **Overview of an exemplary Fog system structure**, that shows nodes with varying sizes, where bigger nodes signal higher computational power. They are classified as *Edge Nodes*, *Standard Nodes* and *Cloud*. *Edge Node* are small computational units distributed at the edge (e.g. Raspberry Pis), *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.

III. Advantages of Auctions

AuctionWhisk makes two contributions: First, it proposes stateless scheduling to mitigate the reliability and latency issues that are inherent with stateful alternatives. Second, by introducing auctions and the market they create, it adds a new layer of logic to consider when deploying functions. In the following two sections, we will discuss advantages and disadvantages of using auctions for scheduling.

A. Statelessness through Auctions

Auctions alleviate the need for centralized state as they can isolate scheduling decisions on each individual node based on a shared algorithm. 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. By isolating scheduling decisions with auctions, expensive synchronization with other nodes or network calls can be avoided, significantly reducing the risk of dependency-induced failures.

B. Minimizing dependencies

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.

C. Improving the Cost-Performance Ratio

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.

IV. Disadvantages of Auctions

A. Complexity through Auctions

However, the competitive nature of auctions may result in economically unjustifiable 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

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

time, the use of these algorithms may shift from an optional optimization to a 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.

B. Volatility and Statelessness in a Distributed System

Keeping to this idea, dynamic bidding algorithms in an auction-based stateless system may cause an issue concerning increased resource overhead and churn. Since auctions are only employed when capacity is exceeded to determine which function should be stored on a node (and which should be evicted), additional overhead and churn can be introduced when functions are moved between nodes continuously in an environment with highly variable load. This effect occurs since there is no component coordinating global load through sophisticated balancing algorithms, but instead load is only propagated upwards towards the Cloud. This can potentially cause issues on two levels, that can amplify when combined. Firstly, moving a function causes increased latency for the application. This is primarily caused, by evicting the function from one node, moving the function data to a new node and starting it there. [4] Secondly, volatile execution patterns are not uncommon in Fog environments. Especially close to the edge, where clients are geographically colocated, there are many factors (e.g., weather, time or events) that can introduce correlated load changes in different applications. [5] For example, observing a deployed application with a low bid: As soon as load increases and the nodes closer to the edge reach maximum resource utilization, the function with a low-bid is evicted and set to be rescheduled on a new node, causing additional resource consumption and a latency increase for the client. This effect can be amplified and cause significant load spikes when such eviction processes happen at the same time, due to correlation in application request variability.

C. Latency Caused by Auctions

Another potential concern is, that although auctions can decrease latency through stateless, dependency-free operations 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 functions 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, that performance is worse than it would have been, routing the request directly to the cloud. That particularly means, that in this scenario the Fog system can put an application at a disadvantage, 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.

[4] Such a first execution (on a new node) is called a coldstart, which consumes additional resources and takes longer, since a new virtualization environment is created.

[5] The closer one moves to the Cloud the less apparent these effects become, as there are more applications – which through uncorrelated load patterns – balance each other out.

V. Applicability

In this section we will summarize our findings from Section III and Section IV to judge AuctionWhisk's general applicability in common Fog Computing and FaaS use-cases in order to determine its potentials and deficiencies in those areas.

A. Summary

AuctionWhisk is able to offer a stateless approach to scheduling functions in a distributed environment. This offers simplicity to the system and reduces operational overhead for the provider, while also giving the client more freedom and control over cost and latency requirements. Furthermore, the approach can limit the effects of dependency-induced failures.

Contrarily, auctions also introduce additional complexity into FaaS, counteracting the paradigm's promise of simplicity. Moreover, auctions can introduce overhead at multiple levels: First, bidding strategies can cause price spikes for function execution which can potentially lead to increased resource overhead and churn. Second, the concept of auctions requires aggregation of function calls in a component that delay request execution. This effect can be amplified at multiple layers by the hierarchical structure and request propagation in AuctionWhisk.

B. Use Cases

In the following, we will only consider use cases that generally benefit from the locality and very low-latency advantages Fog or Edge Computing contribute to fairly determine AuctionWhisk's applicability based on its contributions. For this we will first examine the properties a deployed application on AuctionWhisk should have and then elaborate this on two examples.

A use case that complements AuctionWhisk's approach, should have the necessity for dynamic request priority adjustments or cost-minimization. If this property is not given, the application would likely not benefit much from the advantages of auctions and instead would only have to face the increased complexity they introduce. Furthermore, it should have higher reliability requirements than an application that is normally

[6] Auction latency is here defined to also include serialization, deserialization, IO-operations etc.

deployed in a Fog environment. That is, although the statelessness AuctionWhisk provides increases reliability by reducing cascading failures, it also sacrifices global load balancing that can be beneficial during load spikes to reduce overall latency.

Consequently, a suitable case might be video footage evaluation of events with high crowd-density to determine potential panic dangers and send early warnings. Here, there is a need for low latency in the limited periods where events take place. Furthermore, reliability is important as failure in an upstream component should not cause system failure in any case during the critical time periods. Importantly, the cameras do not have the needed computation power to evaluate the video-material, hence a remote system is required that is optimally edge distributed to avoid separate management at many potential event locations.

Contrarily, a non-suitable use case could be high-scale traffic evaluation to analyze long term driving behavior in different regions or during different times of the day. As there is always traffic, there is no temporal restriction and since the system is only meant to analyze and not to warn there is no low-latency requirement or a necessity for dynamic priority adjustments. Similarly, as monitoring is conducted over long time-periods, reliability is not critical and small outages can be tolerated. More importantly, cost should be predictable in relation to the performance which AuctionWhisk cannot guarantee.

VI. Conclusion

AuctionWhisk adds a new dimension to Fog Computing and the FaaS paradigm by introducing auctions. This extension has the potential to increase reliability in unreliable Fog systems. Moreover, it offers clients of such a system more control over cost in relation to performance and gives them flexibility through bidding strategies. However, auctions also introduce more complexity into the system and counteract the common FaaS promise of an easily useable platform. In addition, by being strictly stateless, auctions do not offer sophisticated function scheduling on a global level which worsen resource utilization and efficiency.

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