Federated Learning for Service Placement in Fog and Edge Computing

Manuel Dworzak and Marcel Großmann and Duy Thanh Le

Computer Networks Group, University of Bamberg
Bamberg, Germany
Email: {firstname.surname}@uni-bamberg.de

***Abstract*—Service orchestration requires enormous attention and is a struggle nowadays. Of course, virtualization provides a base level of abstraction for services to be deployable on a lot of infrastructures. With container virtualization, the trend to migrate applications to a micro-services level in order to be executable in Fog and Edge Computing environments increases manageability and maintenance efforts rapidly. Similarly, network virtualization adds effort to calibrate IP flows for Software-Defined Networks and eventually route it by means of Network Function Virtualization.**

Nevertheless, there are concepts like MAPE-K to support micro-service distribution in next-generation cloud and network environments. We want to explore, how a service distribution can be improved by adopting machine learning concepts for infrastructure or service changes. Therefore, we show how federated machine learning is integrated into a cloud-to-fog-continuum without burdening single nodes.

***Keywords*—Fog Computing; SDN; Orchestration; Federated Learning;**

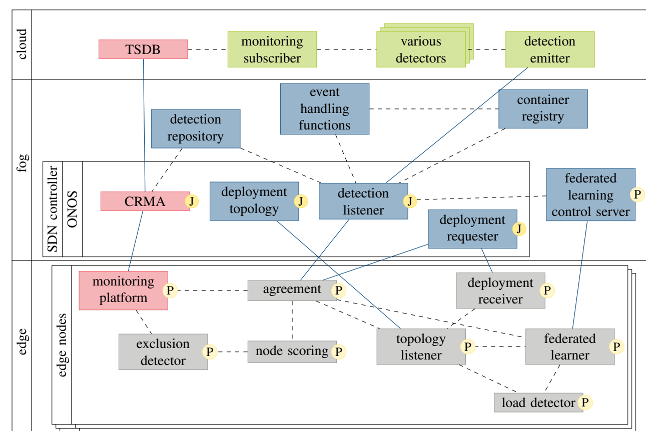


Figure 1. High level architecture [4]

I. Introduction

In the early days of personal computers, when the Internet was not yet around, the question of service orchestration did not exist. Computer users only ran programs located and installed on the computer they were currently using. With the rise of the Internet and global connectivity, this paradigm of local-only shifted. Cloud Computing (CC) constitutes another major shift in which not only the code is stored remotely but also executed on data-center machines far away from the computer user. With this shift, a new question arose: On which server shall the code run? The cloud provider must choose between several servers, ranging from a few to thousands of servers, with very potent hardware. The great advantage of CC, due to these potent servers, is that a non-optimal decision may not lead to significant performance loss as the servers have enough residual capacity in most cases. This non-significant performance loss changed with the upcoming paradigms of Fog and Edge Computing (FEC). The decision changed from thousands of servers to millions or billions of nodes and devices. Additionally, computing devices are not as powerful anymore and differ significantly in their capacities. Consequently, it leads to a more complex decision on which computing device to deploy as the decision variables and the number of devices increase. Furthermore, non-optimal decisions lead to more significant performance loss as some devices may not have the computing capabilities to balance out decision mistakes. So the core question stands: How to optimize service deployment?

At first, we give a short overview of current state-of-the-art research on container orchestration and Autonomic Computing.

IBM introduced a blueprint of Autonomic Computing and its characteristics, such as self-healing, self-configuring, self-protecting, and self-optimizing [5]. They provide the monitoring, analysis, planning and execution over the knowledge base (MAPE-K) reference architecture to build a system that implements the given characteristics.

Casalicchio [1] provides the problem definition and research challenges of autonomic orchestration. They hint at the current research challenges from monitoring, over performance modeling, to adaptation models. Furthermore, they explain that the MAPE-K architecture is the predominant orchestration architecture in Autonomic CC systems.

Costa et al. [2] found that the main challenges in the context of Fog Computing (FC) orchestration are privacy and security, evaluation in real environments, and standardized execution environments. In addition to finding research challenges, their survey states the following research questions: What is the goal of the orchestration, what are the orchestration entities, and what is the orchestration control topology, and which architecture layers shall one consider [2]?

As we will use machine learning (ML) in our architecture,

This document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0):
http://creativecommons.org/licenses/by-sa/4.0 This CC license does not apply to third party material (attributed to another source) in this publication.

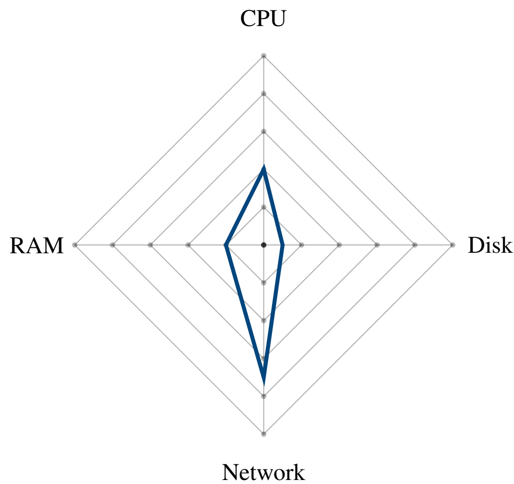


Figure 2. Radar chart for resource estimation

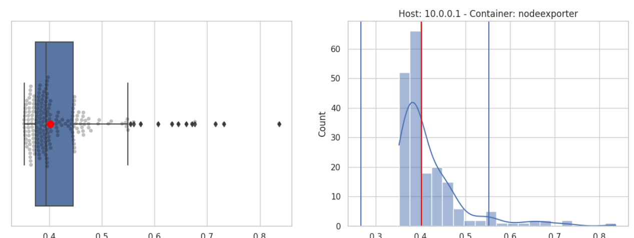
we also want to present studies that research the combination of container orchestration and ML. Naydenov and Ruseva [6] give a broad overview of the domains of ML in container orchestration and the ML approaches and algorithms. Zhong et al. [7] go into more detail with more in-depth literature research about ML algorithms used in container orchestration. Additionally, the authors review each of the ML algorithms and orchestration methods by their characteristics in terms of advantages and disadvantages.

In our previous work we gave insights, how the MAPE-K paradigm is realized for container orchestration, especially in the cloud-to-fog-continuum as shown in the architecture depicted in Figure 1. We proposed a three-layer Autonomic Computing architecture for Docker[1] container deployment for FEC scenarios using Software-Defined Networks (SDNs). Additionally, we added a federated machine learning (FL) concept to ease the decision making process [4]. Now, we show various technologies and tools we used to accomplish our goal, ranging from the modalities of computing paradigms to ML techniques.

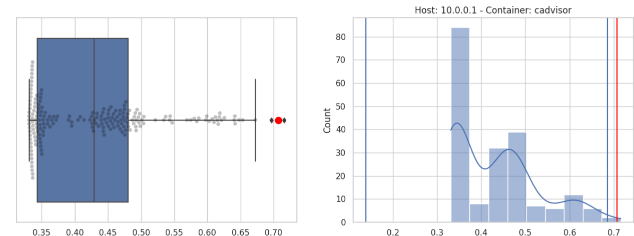
II. Services to Support Decision Making

A. Resource Estimation

We ask application providers to annotate every container deployment request with a radar chart as shown in Figure 2. Hence, we can learn the required resources from the already running containers due to these chart annotations, using FL on each node with each container. For FL, each node needs to know about the network topology such that the nodes can interact with each other, which the *topology listener* provides, which gets this data from the *SDN Controller*. For each deployment request, the *node scoring* module creates a score based on the estimated resource usage. The node scoring process considers factors like already used ports, container names, or resource overload, which the *exclusion detector* verifies. The *deployment receiver* component deploys the container to the highest scored node, which all eligible nodes agreed on by a distributed *agreement* algorithm.



(a) Non-anomaly plot



(b) Anomaly plot

Figure 3. Anomaly detection

B. Load Detection

We avoid any Quality of Experience (QoE) impact on currently running services by running the learning process only, when the system has an overall low resource usage. Contrary, on an overall high resource usage, each container's resource usage information is stored in a node-local database. This data is used for learning by the FL process. Hence, we store in the *load detector* one model for each hierarchy and a global one.

C. Taint Detection

The concept of tainted nodes is taken from Kubernetes[2] and used to prevent nodes with high resource usage from deploying new containers. The *taint detector* receives the tree structure from the *monitoring subscriber* and checks in the tree structure if any node has fulfilled a CPU, memory, or disk load over 60%. If any of these three conditions are satisfied, we consider the node tainted, and the *taint detector* forwards this information to the *detection emitter*.

D. Anomaly Detection

The *anomaly detector* aims to detect anomalies by an *extended isolation forest* for hosts and containers. These forests are relearned every day to be up to date with current usage trends. Whenever the *monitoring subscriber* sends a tree to the *anomaly detector*, it calculates an anomaly score. If it exceeds a host or container-specific threshold, the event is forwarded to the *detection emitter*.

For each container and host, we use the CPU, memory and disk usage, as well as, the network sent and receive rate to generate the extended isolation forest. To avoid the swamping problem and keep the forests small enough for

[1] https://www.docker.com/

[2] https://k3s.io/

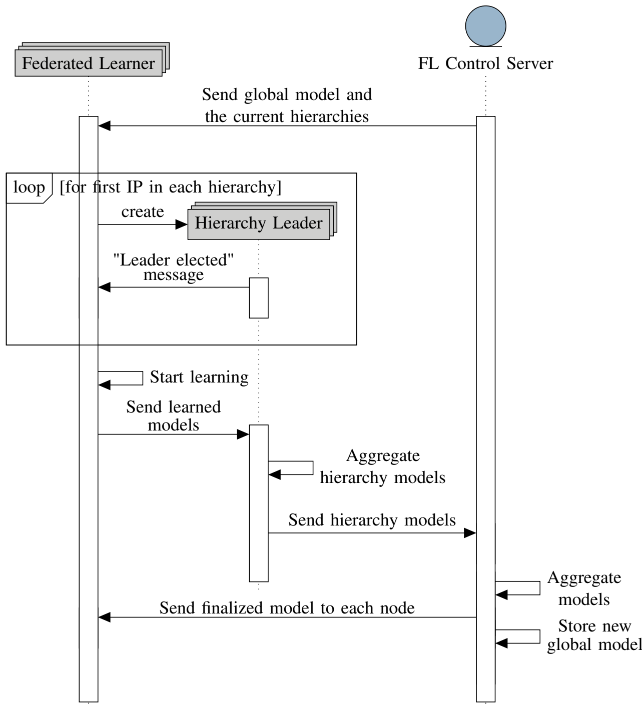


Figure 4. FL workflow

efficient computation of anomaly scores, we only use 1000 data points for each extended isolation forest. We create a boxplot of anomaly scores for these data points, where the whiskers are the threshold to detect anomalies for each isolation forest. In Figure 3 the two blue vertical lines in the right images represent the whiskers, while the red vertical bar on the right images represents the current anomaly score, depicted as a red dot in the left image. We can see in Figure 3a that the red vertical bar or red dot, the current anomaly score, is inside the blue vertical bars or the whiskers of the boxplot. It means we do not consider the current anomaly value as an anomaly. In Figure 3b, we can see what it looks like when detecting an anomaly. The anomaly score exceeds the whiskers of the boxplot.

For every tree the *monitoring subscriber* receives, we calculate the anomaly score and compare it with the whiskers of the isolation forest model. If the anomaly score exceeds the whiskers, an anomaly is detected. However, we also want to detect which kind of anomaly we have which is not supported by isolation forests by default. Hence, we calculate the average and standard deviations for each parameter (CPU, Memory, Disk, Network RX, Network TX) and how many standard deviations the current value is divergent from the average. The *anomaly detector* sends this array with the container name and the host IP to the *detection emitter*. Based on this information, the *event handling functions* component can use different strategies to resolve the anomaly event.

III. Federated Learning

A chart diagram is the basis for the *federated learner* to estimate the container resource usage. Therefore, it uses a local data model. For each hierarchy, an elected leader will aggregate the learned models of its hierarchy. They are sent to the *FL control server*, which is the counterpart that aggregates those models.

We show the workflow for the whole FL process in Figure 4. When an iteration begins, the *FL control server* sends the global model, the hierarchy name, and the list of IP addresses of the current hierarchy to each node. The first IP address is considered the hierarchy leader in this iteration. This way of leader election is simple and efficient. However, it does not give respect to the resources on the node. Once the leaders are elected, they send a "Leader Elected" message to their hierarchies. By receiving the message each node trains the model from its local data. After the *federated learner*s have finished their learning process, the nodes send their models to their hierarchy leader, which aggregates them. Once the merging is complete, all hierarchy leaders send the aggregated models to the *FL control server*, which aggregates all hierarchy models. After this aggregation, the *FL control server* sends the finalized model to each node and stores the global model.

IV. Conclusion

Our architecture is already shown in our previous work [4], which is structured according to the MAPE-K paradigm. Different aspects were additionally split among the network layers of edge, fog, and cloud computing.

Several existing Docker container orchestrators, Kubernetes[2] and Docker Swarm[1], as two notable mentions, use a centralized node scoring process in which a central solver iterates through all nodes for the deployment and scores them individually. Increasing the node count increases the time spent on the scoring process due to more required computations by the central solver. We solved this goal by introducing a decentralized scoring function in which each node scores itself in parallel. A centralized server process only needs to find the node with the highest score.

Taken from Delimitrou and Kozyrakis [3], we know that in 70% of the cases, users overestimate the required resources while 20% underestimate the resources. We give users a radar chart to fill out for resource estimation and then use ML techniques to learn the used resources. This ML approach allows mitigation of the issues that the majority overestimates or underestimates resource requirements as the ML approach learns and adapts its model constantly.

In the future, a critical addition would be moving some detectors to the fog layer. In the current state, if a new machine enters the network and a *new machine detector* suffers from packet loss, this new node will never be used for deployment. Instead of using dedicated detection processes, we could leverage the northbound protocols of Open Network Operating System (ONOS) to get information about the current networking nodes. In the future, more research on which analysis steps are vital to the system is required to move them to the fog layer to reduce possible failure scenarios, as the cloud layer has three single point of failures (SPoFs), and mitigate them further.

References

1. Casalicchio, E.: Autonomic Orchestration of Containers: Problem Definition and Research Challenges. In: Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools. VALUETOOLS'16, 287–290. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Taormina, Italy (2017). DOI: 10.4108/eai.25-10-2016.2266649
2. Costa, B., Bachiega, J., Carvalho, L.R. de, and Araujo, A.P.F.: Orchestration in Fog Computing: A Comprehensive Survey. ACM Comput. Surv. 55(2) (2022). DOI: 10.1145/3486221
3. Delimitrou, C., and Kozyrakis, C.: Quasar: Resource-efficient and qos-aware cluster management. ACM SIGPLAN Notices 49(4), 127–144 (2014)
4. Dworzak, M., Großmann, M., and Le, D.T.: Federated Autonomous Orchestration in Fog Computing Systems. In: Yang, X.-S., Sherratt, S., Dey, N., and Joshi, A. (eds.) Proceedings of Eigth International Congress on Information and Communication Technology, forthcoming. Springer Nature Singapore, Singapore
5. IBM Autonomic Computing: White Paper: An architectural blueprint for autonomic computing. (2005)
6. Naydenov, N., and Ruseva, S.: Combining Container Orchestration and Machine Learning in the Cloud: a Systematic Mapping Study. In: 2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH), pp. 1–6 (2022)
7. Zhong, Z., Xu, M., Rodriguez, M.A., Xu, C., and Buyya, R.: Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions. ACM Computing Surveys (CSUR) (2021)