Computer Networks Group
University of Bamberg, Germany

Federated Learning for Service Placement in Fog and Edge Computing

WueWoWas 2023

June 30, 2023

Manuel Dworzak, Marcel Großmann, Duy Thanh Le

Outline

Orchestration & Autonomic Computing

Orchestration

"Service Orchestration refers to the composition of system components to support the Cloud Providers' activities in arrangement, coordination and management of computing resources to provide Cloud services to the Cloud Consumers" [1].

Autonomic Computing

Autonomic Computing is a property of software systems that fulfill the following four properties, taken from [2].

- **Self-configuring**: The system shall dynamically configure itself to adapt to changing environments
- **Self-healing**: The system shall detect and react to errors
- **Self-optimizing**: The system shall optimally assign resources to improve overall system performance
- **Self-protection**: The system shall detect and react to hostile behavior

MAPE-K

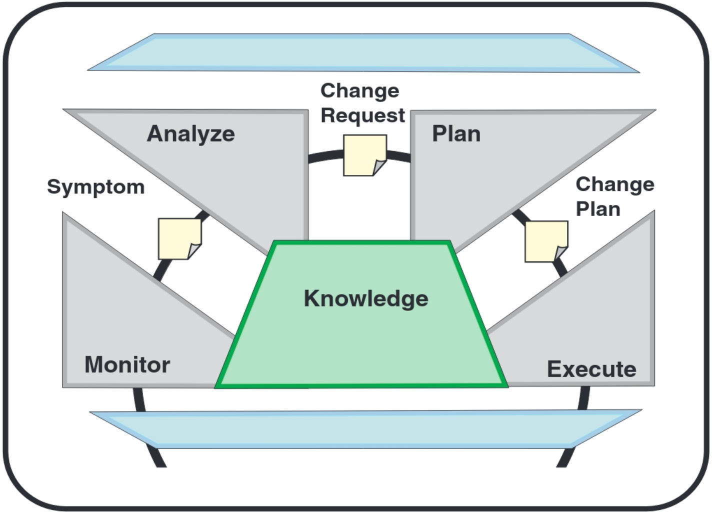


Figure 1: MAPE-K Architecture [2]

Computing Paradigms

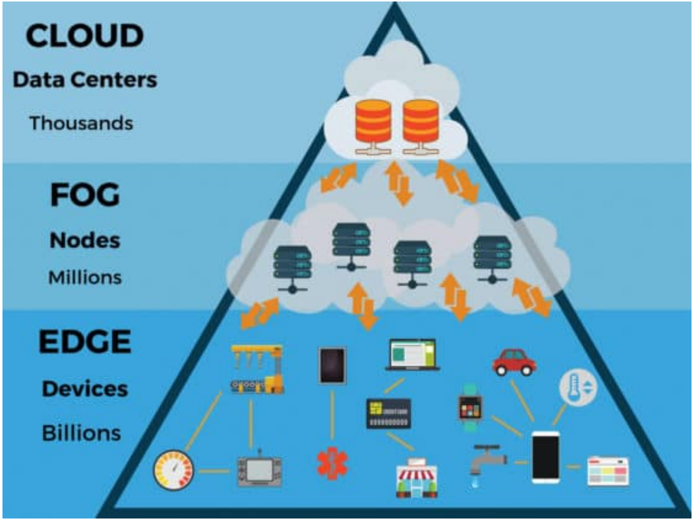


Figure 2: Computing Paradigms [3]

Learning Principles

The approximation theorem states that a feed-forward network with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any function from one finite-dimensional space to another with any desired non-zero amount of error provided that the network is given enough hidden units. [cf. 4]

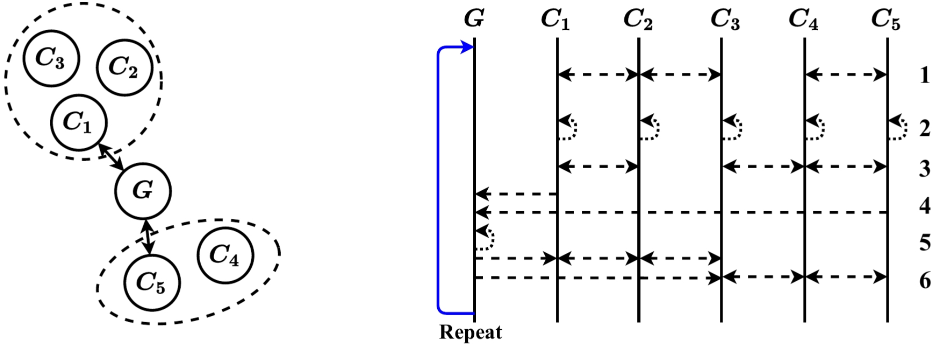
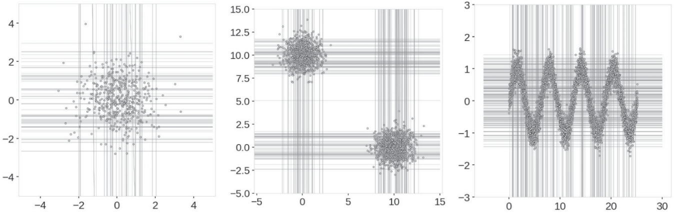


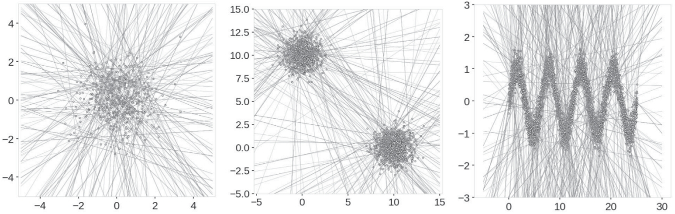
Figure 3: Federated Peer to Peer [5]

Isolation Forests



(a) Single blob (b) Multiple Blobs (c) Sinusoidal

Figure 4: Isolation Forest Bias [6]



(a) Single blob (b) Multiple Blobs (c) Sinusoid

Figure 5: Extended Isolation Forest Bias [6]

Comparison of Traditional and Software-Defined Networking

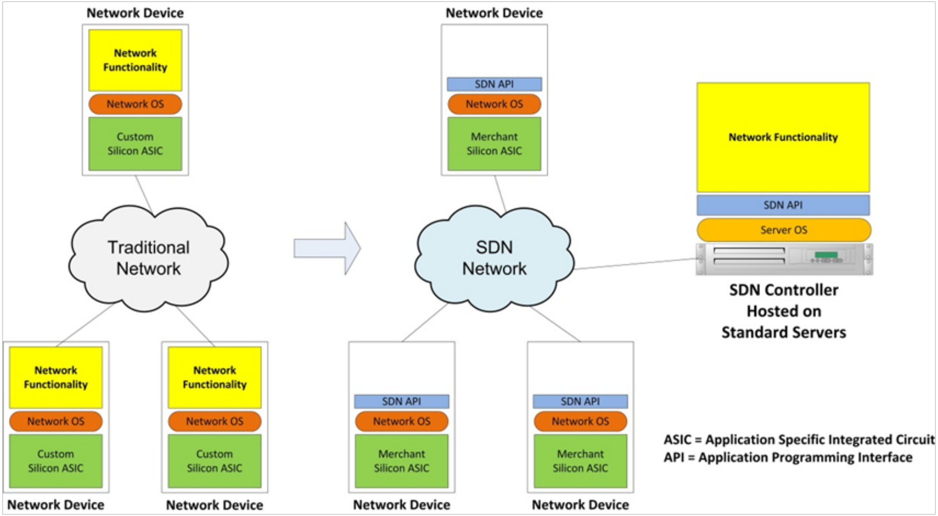


Figure 6: Difference Traditional and Software Defined Networking [7]

Outline

Solution Space

Dynamic Event Handling

Dynamic Event Handling requires strong monitoring and data analysis processing.

Kubernetes' Scalability Considerations

The Kubernetes Scheduler scores fewer nodes when too many nodes are available: "Kubernetes calculates a figure using a linear formula that yields 50% for a 100-node cluster and yields 10% for a 5000-node cluster"[1].

Support for Node Scalability

Our approach implements decentralized scoring such that every nodes scores itself to avoid running in such performance issues.

[1] `https://kubernetes.io/docs/concepts/scheduling-eviction/scheduler-perf-tuning/#default-threshold`

Intelligent Container Resource Estimation

A user enters the following three pieces of information for new deployments:

- A **hierarchy** for the application deployment
- A filled out chart rating about resource usage for each container
- The deployment configuration with the Docker container configuration

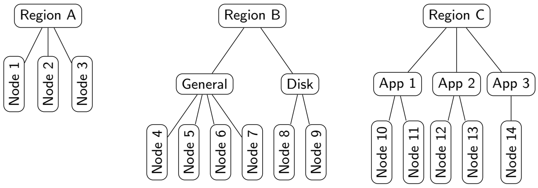


Figure 7: Hierachy of nodes for filtering

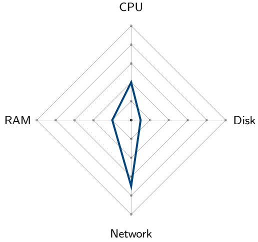
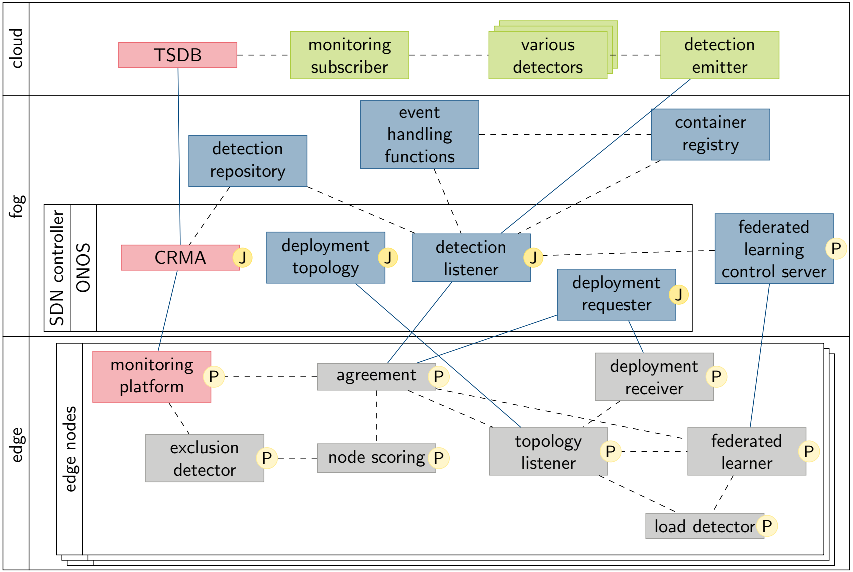


Figure 8: Radar chart for resource estimation

Architecture



Cloud Layer

Load Detection

The **machine learning** process aims to approximate a function that reflects the resource usage at the given time. A two layer neural network for function approximation receives the timestamp as input and returns the expected $score_{resources}$ value, which averages the CPU and memory usage (cf. Equation 2.1).

$$score_{resources} = \frac{cpu\_usage + mem\_usage}{2} \tag{2.1}$$

Cloud Layer

Taint Detection

Concept of tainted nodes is taken from Kubernetes [8].
Taint detector checks if any node fulfills one of the following conditions:

- CPU Load over 60%
- Memory Load over 60%
- Disk Load over 60%

For each container and host, we use the following parameter to generate the Extended Isolation Forest:

- CPU usage
- Memory usage
- Disk usage
- Network RX/TX rates

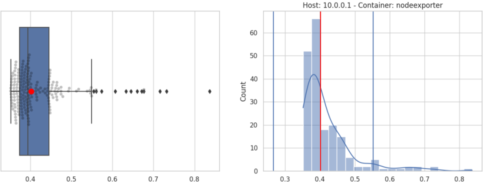


Figure 9: Non-anomaly plot

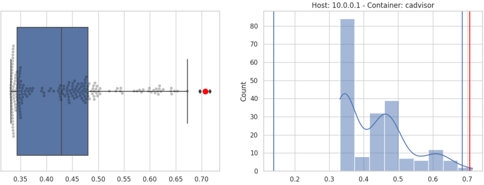
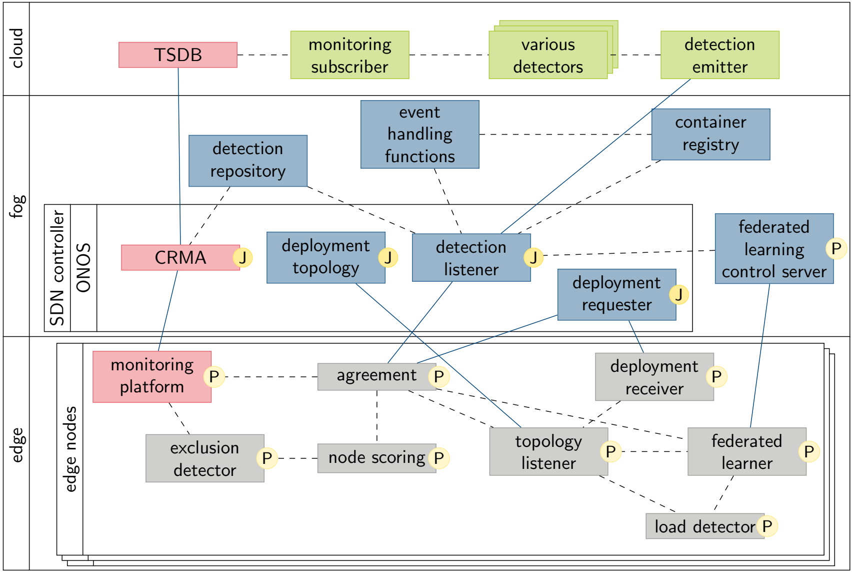


Figure 10: Anomaly plot

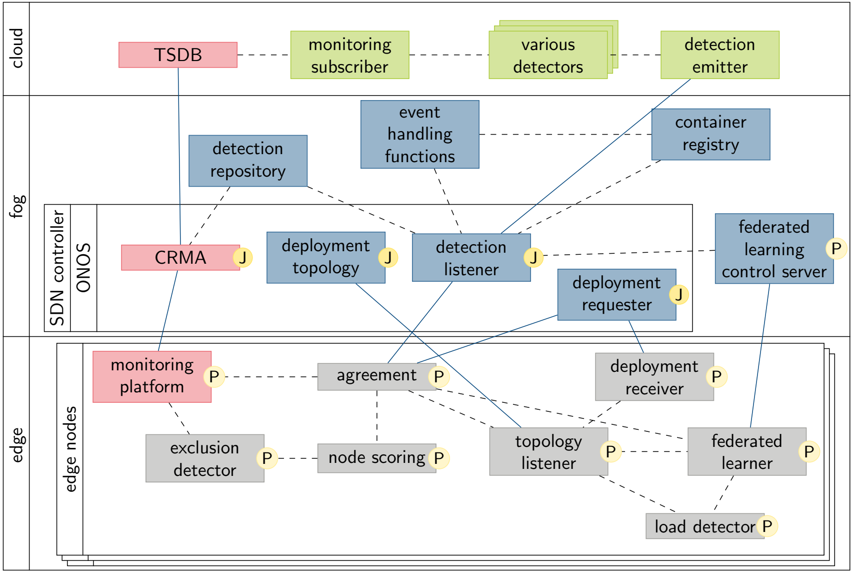
Architecture



Fog Layer

The *detection listener* is the *P* in our MAPE-K architecture. It reacts to the analysis done by the cloud layer by looking up event handlers in the *detection repository*. If we specify an event handler for the given scenario, the *detection listener* makes an HTTP call to the event handler URL.
The *deployment requester* deploys new containers and interacts with edge nodes for agreement and scoring.
It needs to know the hierarchy for the application deployment with its container configurations.
With this resource estimation a service can be deployed on an appropriate node.

Architecture

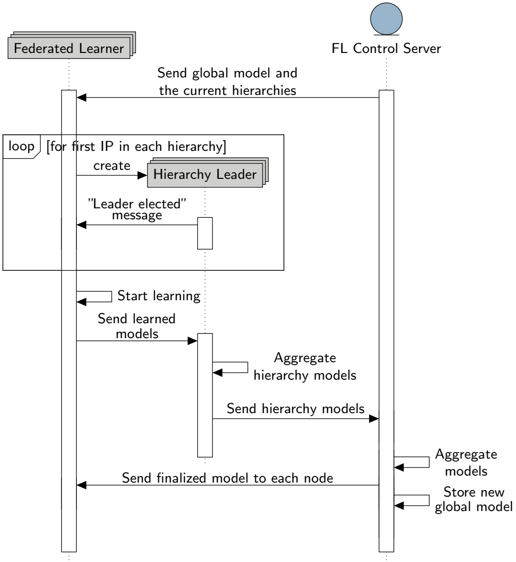


Edge Layer

As our **target devices** are **edge devices**, we also want to add factors like disk usage and network usage to the set of metrics M and calculate their utilization.
Combining the exploitation factor f, where f is in $[0,1]$, with the resource utilization equations, we finally calculate the overall node utilization score by Equation 2.2.

$$\begin{aligned} M &= \{cpu, mem, disk, net\} \\ S_x &= \frac{T_x - S_{req\_x} - f * P_{req\_x}}{T_x} \qquad for\ x \in M \\ score_1 &= \frac{\sum_{x \in M} S_x}{|M|} \end{aligned} \tag{2.2}$$

Federated Learning Workflow



Outline

Approximation of the Load Detection I

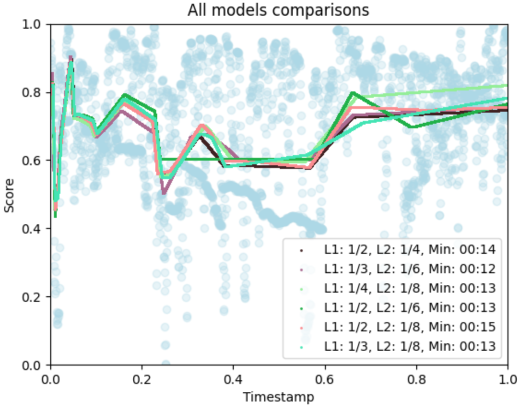


Figure 11: Comparison of different approximations through the load detection function

Approximation of the Load Detection II

Explanations

- Blue dots are the score values over time, the orange line is the approximated function.
- Layer 1 and 2 node count, given as a fraction, like $1/2$ or $1/4$
- Minimum load time is between 00:13 and 00:15 in this scenario.
- Difference between the configurations is mostly insignificant.
- The size of the layers does not make a significant difference in finding the minimum value

Outline

Reflection on Research Goals

Conclusion

- Evaluate the MAPE-K architecture for container orchestration
- Dynamic event handling
- Support for node scalability
- Intelligent container resource estimation

Future Work

- Container migration
- Constraints and goals (similar to DRAGON [9])
- Privacy and security
- Testing in real production environment

References

[1] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, D. Leaf *et al.*, "Nist cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, pp. 1–28, 2011.

[2] I. A. Computing, "White paper: An architectural blueprint for autonomic computing," 2005.

[3] "Edge and fog computing: Their practical uses," https://iot.electronicsforu.com/content/tech-trends/edge-and-fog-computing-practical-uses/, accessed: 2021-12-01.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[5] L. Chou, Z. Liu, Z. Wang, and A. Shrivastava, "Efficient and less centralized federated learning," in *Machine Learning and Knowledge Discovery in Databases. Research Track*, N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 772–787.

[6] S. Hariri, M. C. Kind, and R. J. Brunner, "Extended isolation forest," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1479–1489, 2019.

[7] "Sdn vs traditional networking: Which leads the way?" https://www.chinacablesbuy.com/sdn-vs-traditional-networking-which-leads-the-way.html, accessed: 2022-05-20.

[8] "Kubernetes taint based evictions," https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#taint-based-evictions, accessed: 2021-12-28.

[9] G. Castellano, F. Esposito, and F. Risso, "A service-defined approach for orchestration of heterogeneous applications in cloud/edge platforms," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1404–1418, 2019.

Questions ?

Manuel Dworzak
manuel.dworzak@uni-bamberg.de
Marcel Großmann
marcel.grossmann@uni-bamberg.de
Duy Thanh Le
duy-thanh.le@uni-bamberg.de