Nimble: QoS-Aware Resource Management for Edge-Assisted Microservice Environments

Amit Samanta
University of Utah

Abstract

We propose Nimble, a QoS-aware resource management framework for edge-assisted microservice environments. We build a preliminary prototype of Nimble and show its applicability in terms of resource utilization and execution latency for microservices in a small-scale testbed setup.

CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

Microservice environments, resource management, QoS.

ACM Reference Format:
Amit Samanta. 2023. Nimble: QoS-Aware Resource Management for Edge-Assisted Microservice Environments. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3570361.3615738

1 Introduction

With the increase number of Internet-of-Things (IoT) applications [4, 9, 13, 15, 22, 24] and programmability of cloud platforms, most of the applications are uploaded from mobile to cloud platforms. However, uploading to cloud platforms incur high latency due to limited effective bandwidth and link failures, therefore it fails to provide Quality-of-Service (QoS) requirements to IoT applications, such as video streaming and virtual reality [6]. Such problems are resolved by introducing the edge computing platforms [8, 10, 23], it basically minimizes the network latency and optimizes the available resources. These days, edge platforms have adopted microservice-oriented frameworks [3], where the applications are decomposed into small and independent microservices to provide better performance to edge-assisted users [7]. The microservice-oriented framework [2, 11, 12, 17, 18, 21] provides better parallelism and efficiency to services on the edge-assisted servers using standard containerized technology to enable easy programmability, minimal deployment efforts and low cost for their executions. The containerized framework of microservice instances is considered to be one of the important example of edge platforms. Here, the IoT applications are decomposed into a significant number of microservices and their instances are executed on the edge-assisted servers to optimize the performance.

The main property of such microservice-oriented edge platforms is to execute the microservice instances efficiently placed on the containers of edge-assisted servers. On edge platforms, the microservices are mostly presented in terms of Function-as-a-Service. The edge-assisted containers provide heterogeneous resources [20] (CPU, memory, storage, network) to microservice instances for efficient execution and be in the service of incoming edge-assisted users while maintaining the minimal resource provisioning price. For most of the edge platforms providing minimal pricing and latency to edge-assisted users comes among the highest priorities. The incurred provisioning price of microservices comes from the operational price of edge-assisted servers and managing/scaling the containers, which is mainly determined by the fractures and the variety of containers serving several kinds of microservices. Similarly, the edge platforms need to spawn up brand new microservice instances on the edge-assisted servers, which includes migration of container images, starting and connecting them to appropriate instances. This basically incurs the deployment price, which is determined based on the price of running containers for a particularity duration. Hence, it needs to be optimized, by circumventing recurrent deployment and expulsion of microservice instances. This work tackles the problem of providing adequate amount of resources to microservices for edge platforms. Given the multiple containers are running in edge-assisted servers, the main question is: how to allocate resource fairly and adequately to microservices to accelerate their execution with fluctuating workload changes and achieve better QoS for edge-assisted users. The important challenge in designing such solution is to come up with an online algorithm to accomplish the upper-mentioned objectives in the presence of rapid workload changes significantly over time. We present Nimble, a QoS-aware resource management framework for microservices, which provides better resource utilization and minimal latency.

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10...$15.00
https://doi.org/10.1145/3570361.3615738

2 QoS-Aware Resource Management for Microservice Environments

The main objective of QoS-aware resource management framework is to provide adequate amount of resources to microservices and place them in the edge-assisted containers to optimize the resource utilization of microservice environments over time slot T. We formulate the scaling factors: (a) container placement factor $\mathcal{I}_i(t)$, it represents the number of microservice i instances placed on edge-assisted server s at time t; (b) resource management variable $\mathcal{J}_i(t)$, it represents the unit amount of resources available in a container for microservice i on edge-assisted server s at time t. The total resource management price are discussed as:

Price Estimation: To approximate the total resource management price, we design a price estimation framework.

- *Functional Price:* Suppose α_i presents the unit price of running and offloading an instance of microservice i to edge-assisted servers at time t, basically ascribed as the functional price. Therefore, the total functional price is described as:

$$\mathcal{P}_{fun} = \sum_{t\in|T|}\sum_{s\in|S|}\sum_{i\in|M|} \alpha_i \mathcal{I}_i(t) \tag{1}$$

- *Placement Price:* While placing a brand new instance of microservice i, the framework needs to migrate the image of microservices to new containers on the edge-assisted servers. We consider the unit placement price β_i for microservices. Therefore, the overall placement price is described as:

$$\mathcal{P}_{pl} = \sum_{t\in|T|}\sum_{s\in|S|}\sum_{i\in|M|} \beta_i \mathcal{I}_{i,new}(t) \tag{2}$$

  where $\mathcal{I}_{i,new}(t)$ indicates the newly added instance of microservice i on edge-assisted server s at time t.

- *Execution Price:* Once the placement, the microservices are allocated container resources on the edge-assisted servers for execution. We consider the unit resource management price ϑ_i for microservices. The overall resource management price is described as:

$$\mathcal{P}_{re} = \sum_{t\in|T|}\sum_{v\in|V|}\sum_{i\in|S|} \mathcal{W}\vartheta_i \mathcal{J}_i(t) \tag{3}$$

  where $\mathcal{W}$ indicates number of microservices present in a container. The aim is to optimize total price $\mathcal{P}_{to} = \mathcal{P}_{fun} + \mathcal{P}_{pl} + \mathcal{P}_{re}$.

Resource Management Framework: Following the price estimation, we expressed the QoS-aware resource management optimization problem for microservices. The problem directs to allocate resources to microservices while providing QoS to edge-assisted users, as microservices require adequate amount of container resources for their execution. Thus, we design a QoS-aware resource management optimization framework considering total management cost and resource utilization. We construct a joint resource optimization framework for microservices. Therefore, we have,

$$\text{Min} \sum_{t\in|T|}\sum_{s\in|S|}\sum_{i\in|M|} \Bigg[\overbrace{\mathcal{R}^t_{\mathcal{U},i}}^{\text{resource utility}} = \underbrace{\chi_1 \mathcal{P}_{to}}_{\text{total price}} - \chi_2 \underbrace{\frac{Q^t_i}{Q^t_{th}}}_{\text{QoS factor}} \Bigg], \tag{4}$$

$$\text{subject to} \quad \chi_1, \chi_2 = \{0, 1\}, \tag{5}$$

$$\mathcal{P}_{to} \le \mathcal{P}^{th}_{to}, \forall i \in |M|, \tag{6}$$

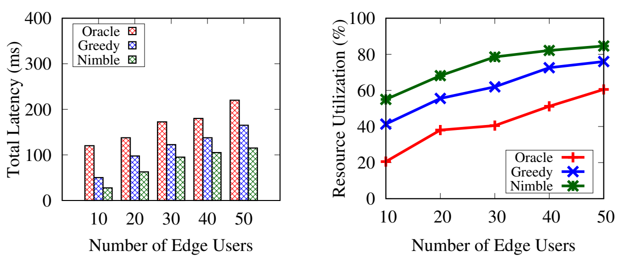
$$Q^t_i \ge Q^t_{th}, \forall i \in |M|, t \in |T|, \tag{7}$$

where $|S|$ and $|M|$ indicate the number of microservices and edge-assisted servers, $\mathcal{P}^{th}_{to}$ indicates the threshold management price, Q^t_i and Q^t_{th} indicate the measured and threshold QoS factor. (4) indicates the joint optimization problem for microservices. The scaling factors are discussed in (5). (6) indicates that the total cost $\mathcal{P}_{to}$ needs to be lesser than threshold cost $\mathcal{P}^{th}_{to}$. (7) indicates that the measured QoS factor Q^t_i needs to be lesser than the threshold QoS factor Q^t_{th}. Following the properties of linear optimization problem, we designed the heuristics for Nimble and solve the integer liner program (ILP). It provides minimal computational complexity than other methods. We compare Nimble with other methods - Oracle (requires prior microservice information) and greedy solution.

3 Preliminary Results

We design an initial small-scale prototype to emulate the Nimble framework and edge platforms by software wrapping [7, 8, 14, 19]. In the current form, it shows some initial performance advantages, but it may not be able to provide the full functionalities of Nimble. The evaluation was done on a 9 server m510 cluster of Intel Xeon D-1548, 2.0 GHz CPU and 64 GB ECC Memory with DDR4 RAM on the CloudLab testbed. We consider a client/server configuration to generate the microservices from hotel reservation application and estimate the resource utilization. The hotel reservation application is running on a single server and offloading the microservices to other 8 edge-assisted servers to execute the microservices effectively. The edge-assisted servers dwell on Ubuntu 20.04 with Linux Kernel 4.15 and equipped with Dual-port Mellanox ConnectX-3 10 GB NIC. For the generation of microservice workloads, the Poisson distribution is considered with mean between 0 and 0.5.

Experimental Setup. For the experiments, 50 edge-assisted users are considered and they are equipped with IoT devices running hotel reservation application [1]. The compute power of IoT device is 0.7 GHz. The backhaul latency is setup to be 0.0001 sec/KB [5, 25]. The total time to offload and place the microservices to edge-assited servers are arbitrarily originated between 25 and 50 ms. We vary $\mathbb{P}_{tot}$ within $[20, 60]$. The intermediate data generated between microservices varies within the range of 1 and 10 MB. The latency of IoT devices is considered to be within 0.5-1s.



(a) Total Latency (b) Resource Utilization

Figure 1: Latency and resource utilization of Nimble.

Discussion. Fig. 1(a) represents the total latency (i.e., offloading, placing and executing) incurred by microservices for hotel reservation application. The QoS-aware resource management allows efficient execution of microservices with Nimble, as Nimble provides fair amount of resources to microservices. We also measure the resource utilization of microservices. Nimble assists the edge-assisted users to place and offload the microservices optimally, which allows them to efficiently use the allocated resources and optimize the total resource management price. Therefore, Nimble provides higher resource utilization in Fig. 1(b). Nimble is compared with the existing methods Greedy and Oracle. We observe that Nimble surpasses the other methods.

4 Future Directions

In future, we would like to evaluate Nimble with large-scale experiments to explore it's full functionality. The main idea is to complete the full implementation of Nimble with distributed edge platforms with a large-scale CloudLab cluster. We would like to explore other microservice applications such as Social Network. We would also like to propose an auto-tuning framework for microservices [16] by enforcing dynamic re-configurations of applications.

References

[1] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *ACM ASPLOS*. 3–18.

[2] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2020. Unveiling the hardware and software implications of microservices in cloud and edge systems. *IEEE Micro* 40, 3 (2020), 10–19.

[3] Zhipeng Jia and Emmett Witchel. 2021. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *ACM ASPLOS*. 152–166.

[4] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *ACM ASPLOS*. 615–629.

[5] Xiaoyan Kui, Amit Samanta, Xiangming Zhu, Shigeng Zhang, Yong Li, and Pan Hui. 2018. Energy-aware temporal reachability graphs for time-varying mobile opportunistic networks. *IEEE TVT* 67, 10 (2018), 9831–9844.

[6] Qiang Liu and Tao Han. 2018. DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing. In *IEEE ICNP*. 1–11.

[7] Amit Samanta and Zheng Chang. 2019. Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint. *IEEE IoTJ* 6, 2 (2019), 3864–3872.

[8] Amit Samanta, Zheng Chang, and Zhu Han. 2018. Latency-Oblivious Distributed Task Scheduling for Mobile Edge Computing. In *IEEE GLOBECOM*. 1–7.

[9] Amit Samanta, Xinlei Chen, and Yong Li. 2019. Poster: FlexDP-Flexible Data Plane for ENFV. In *ACM MobiCom*.

[10] Amit Samanta, Flavio Esposito, and Tri Gia Nguyen. 2021. Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty. *IEEE IoTJ* 8, 23 (2021), 16963–16971.

[11] Amit Samanta, Flavio Esposito, and Tri Gia Nguyen. 2023. ASAP: Adaptive and Scalable Microservice Provisioning for Edge-IoT Networks. In *IEEE/IFIP WONS*. 80–87.

[12] Amit Samanta, Lei Jiao, Max Mühlhäuser, and Lin Wang. 2019. Incentivizing Microservices for Online Resource Sharing in Edge Clouds. In *IEEE ICDCS*. 420–430.

[13] Amit Samanta and Yong Li. 2017. DeServE: Delay-agnostic Service Offloading in Mobile Edge Clouds: Poster. In *ACM/IEEE SEC*. Article 24, 2 pages.

[14] Amit Samanta and Yong Li. 2018. Poster: Latency-Oblivious Incentive Service Offloading in Mobile Edge Computing. In *ACM/IEEE SEC*.

[15] Amit Samanta and Yong Li. 2018. Time-to-Think: Optimal Economic Considerations in Mobile Edge Computing: Poster. In *IEEE INFOCOM WKSHPS*. 1–2.

[16] Amit Samanta and Yong Li. 2019. Cost-effective microservice scaling at edge: Poster. In *ACM/IEEE SEC*. 326–328.

[17] Amit Samanta, Yong Li, and Flavio Esposito. 2019. Battle of Microservices: Towards Latency-Optimal Heuristic Scheduling for Edge Computing. In *IEEE NetSoft*. 223–227.

[18] Amit Samanta, Tri Gia Nguyen, Thao Ha, and Shahid Mumtaz. 2022. Distributed resource distribution and offloading for resource-agnostic microservices in industrial iot. *IEEE TVT* 72, 1 (2022), 1184–1195.

[19] Amit Samanta, Quoc-Viet Pham, Nhu-Ngoc Dao, Ammar Muthanna, and Sungrae Cho. 2023. mISO: Incentivizing Demand-Agnostic Microservices for Edge-Enabled IoT Networks. *IEEE TSC* (2023).

[20] Amit Samanta and Ryan Stutsman. 2023. A Case of Multi-Resource Fairness for Serverless Workflows (Work In Progress Paper). In *ACM/SPEC ICPE*. 45–50.

[21] Amit Samanta and Jianhua Tang. 2020. Dyme: Dynamic microservice scheduling in edge computing enabled IoT. *IEEE IoTJ* 7, 7 (2020), 6164–6174.

[22] Dongzhu Xu et al. 2022. Tutti: coupling 5G RAN and mobile edge computing for latency-critical video analytics. In *ACM MobiCom*. 729–742.

[23] Dianlei Xu, Amit Samanta, Yong Li, Manzoor Ahmed, Jianbo Li, and Pan Hui. 2019. Network Coding for Data Delivery in Caching at Edge: Concept, Model, and Algorithms. *IEEE TVT* (2019).

[24] Shuochao Yao et al. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *ACM MobiSys*. 476–488.

[25] K. Zhang et al. 2016. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access* (2016).