A Middleware-Based Approach for Latency-Sensitive Service Provisioning in IoT with End-Edge Cooperation

Canlong Sun[1(✉)], Ting Li[1], Zihao Wu[1], and Cong Li[2]

[1] The 705 Research Institute of China State Shipbuilding Corporation Limited, Kunming, Yunnan, China
362028257@qq.com

[2] Xi'an Institute of Optics and Precision Mechanics, CAS, Xian, Shanxi, China

Abstract. As modern mobile applications have become more and more complex, mobile edge computing brings IT services and computing resources to the edge of mobile networks to full fill various computing and application requirements. Considering that mobile devices may not always have adequate hardware conditions, computation offloading, which can help devices take full advantage of extra computing resources, has reached a broad audience in the edge environments. However, due to the limited storage space of edge servers, it is very difficult to manage services in middleware. Therefore, in the edge computing environment, how to deal with a large amount of data from different edge nodes in the middleware is very important. In this paper, we regard an approach about improving quality of sensitive data for middleware on edge environments. We have evaluated our approaches on a real-world environment. The results demonstrate that our approach can effectively reduce the response time.

Keywords: Edge Computing · Middleware · Latency-Sensitive Service · IoT

1 Introduction

With the popularity of the continuous development of the Internet of Things (IoT) technology, smart mobile devices (e.g., smart phones, tablet PCs, smart home appliances, etc.) have played a increasing important role in people's life [1, 2]. Along with the rapid development of the IoT, the concurrent huge network traffic also arises challenging issues to deliver efficient IoT services with diverse Quality of Service (QoS). Especially, the enormous network traffic load often causes severe network congestion, which eventually impacts the service latency (e.g., response delay). In the field of IoT applications, as IoT nodes usually need to frequently send/receive data to/from the core network server, it greatly increases the load on the central network [3]. To solve the problem of limited terminal resources, mobile edge computing [4, 5] technology offers a plausible solution to empower cellular networks and deliver high bandwidth, low latency, and improved QoS for diverse IoT applications, by placing part of cloud resources (e.g., computing,

R. Li et al. (Eds.): MOBILWARE 2022, LNICST 507, pp. 3–12, 2023.
https://doi.org/10.1007/978-3-031-34497-8_1

storage, and networking capacities) within the edge of radio access network (RAN). As edge computing servers are closer to mobile user terminals, compared with the remote cloud server, the computationally intensive and latency-sensitive applications can be supported by mobile edge computing. Besides, by migrating computing tasks from end device with limited resources to the edge of network, a large portion of mobile traffic will be diverted to edge servers. Thereby, the service-related data can be placed on edge servers to minimize the latency in users' data retrieval [6]. This is especially important for latency-sensitive applications, e.g., gaming, navigation, augmented reality, etc. Popular service often accounts for a large percentage of the mobile traffic data over the internet. Thus, caching popular service as data on edge servers can significantly reduce the traffic load on the internet backbone. According to [7], it is expected to reduce mobile traffic data by about 35% in IoT via the mobile edge computing paradigm.

In this paper, we propose a novel mobile edge computing-based middleware (MDS) approach in IoT to improve the QoS for sensitive data in edge environments. Specifically, we design a novel collaborative service mitigation framework for IoT in the edge environment with MDS. Then, we theoretically formulated the service latency model under MDS in end-edge computing environment, where part of task data is offloaded to an appropriate edge node via the MDS and the other remain to be processed locally. Additionally, we devise an algorithm to effectively sort the processing order of the data arriving in the middleware. Finally, we carry out extensive real-world environments to evaluate the feasibility and effectiveness of the proposed approach in terms of service delay, in comparison with conventional approaches.

The rest of this paper includes the following sections. Section 2 reviews the related work. Section 3 describes system model. Section 3 introduces our proposed MDS approach of replaceable services. Section 4 presents our experimental environment and gives the analysis of experimental results, followed by Sect. 5 that concludes our work and gives the future work.

2 Related Works

The author in [8] propose a novel framework called SpeCH and introduce a new paradigm for partitioning a set of data-items into geo-distributed clouds. In [14], the authors propose a data-centric programming model called Fog Function is proposed, which uses the underlying orchestration mechanism. The authors in [9] propose an approach which is based on a QoS-aware meta orchestration modelling of a given pipeline and an orchestration builder generating deployable Edge-specific orchestrations. The authors in [15] propose an algorithm that divides tasks into four types in real time and then offloads them cooperatively. In [16], the authors propose a Data-intensive Service Edge deployment scheme based on Genetic Algorithm (DSEGA). The authors in [10] propose an edge-enabled federated learning framework for smart grid to deliver intelligent power services for individuals. The authors in [12] develop a novel blockchain system for private data audit in IoT with edge-cloud cooperation. These viewpoints are all based on edge computing or fog computing. They only consider edge nodes and do not consider middleware well. Moreover, data intensive services often transfer large amounts of data over a wide Area network (WAN) because of their scalable fault tolerant protocols

[13]. The authors in the [9] characterize the network stability region and design the first throughput-optimal control policy that coordinates processing and routing decisions for both live and static data streams. The work in [11] proposes an MA-based approach to solving the problem of distributed DWSC in an effective and efficient manner. In particular, the authors in [11] develop an MA approach that combines e-commerce and local search technology with service distance. However, most of these views start from the hardware performance of each node, without considering the thread problem.

3 System Model

3.1 Execution Time of Data Services Under Local Computing

Let $\mathbb{I} = \{1, \ldots, i, \ldots, I\}$ be the set of IoT devices. For the computation task Φ, its execution time of data processing at the local IoT device $i \in \mathbb{I}$ can be computed as:

$$et_{local} = \frac{\alpha_i D_i}{f_i}, \tag{1}$$

where $N_i = \alpha_i D_i$ denotes the number of required CPU cycles to complete the computation task Φ. Here, $\alpha_i (\alpha_i > 0)$ is a parameter related to the computation complexity of the computation task Φ. D_i is the size of the computation task Φ. f_i is the CPU frequency of the IoT device i, which also indicates the local computing resource available for task Φ. Moreover, the corresponding energy consumption in executing task Φ can be computed as:

$$E_{local} = \chi_i (f_i)^3 et_{local} = \chi_i \alpha_i D_i (f_i)^2, \tag{2}$$

where χ_i is the CPU capacitance-related parameter of IoT device i.

3.2 Execution Time of Data Services Under Edge Computing

Let $\mathbb{J} = \{1, \ldots, j, \ldots, J\}$ be the set of edge nodes in the IoT. Under edge computing environment, considering the existence of MDS, the execution time consists of the data transmission latency to the MDS and the edge nodes, the data processing time on edge servers, and the synchronization latency between the MDS and the edge servers. According to [18], the time of downloading services data at edge node $j \in \mathbb{J}$ can be computed as:

$$dt_{edge} = \frac{d_v}{Tx_v} + \frac{wl_c}{c_c} + Q_c + \frac{d_o}{Tx_o}, \tag{3}$$

where Tx_v and Tx_o are the data transmission rate (in Kbps) at the location where the middleware starts to transmitting to each other data about the service c to/from an edge node j, respectively. To calculate Tx_i and Tx_o, We first need to find the location of the middleware through the trajectory model of the middleware, and then use the signal strength of its location to determine its transmission speed. wl_c is the middleware

workload, d_v and d_o mean the granularity of the data. Let $\rho \in (0, 1)$ be the output/input ratio of the task Φ. We have

$$\rho = \frac{d_v}{d_o}. \tag{4}$$

c_c is the CPU capacity (in MIPS) for the middleware. Q_c is the waiting time that edge service c stays in a queue to be executed in the middleware. Moreover, the execution time of data processing at the edge node $j \in \mathbb{J}$ can be calculated as:

$$et_{edge} = \frac{\alpha_i D_i}{f_j}, \tag{5}$$

where f_i is the CPU frequency of the edge node j, which indicates its available computing resource to process the task Φ. Then, the overall service latency can be computed as the sum of the data downloading delay and data processing delay, i.e.,

$$t_{edge} = dt_{edge} + et_{edge} = \frac{d_{i,j}}{Tx_{i,j}} + \frac{wl_c}{c_c} + Q_c + \frac{d_{j,i}}{Tx_{j,i}} + \frac{\alpha_i D_i}{f_j}. \tag{6}$$

The energy consumption in data transmission from the IoT device to the edge node is $E_{trans} = \frac{d_v}{Tx_v} P_i$. Here, P_i is the transmit power of IoT device i. Typically, compared with the original input size of the task, the output size of task (i.e., the task processing result) can be usually negligible. Thereby, the energy consumption of transmisitting the processed task data is neglected. As such, the overall energy consumption in executing task Φ can be computed as:

$$E_{edge} = \frac{d_v}{Tx_v} P_i + \chi_j \left(f_j\right)^3 et_{edge} = \frac{d_v}{Tx_v} P_i + \chi_j \alpha_i D_i \left(f_j\right)^2, \tag{7}$$

where χ_j is the CPU capacitance-related parameter of edge node j.

4 The Proposed Service Provisioning Approach with MDS

4.1 Execution Time of Data Services Under End-Edge Computing with MDS

In practical IoT applications, part of the task can be processed locally while the other part of the data can be effectively offloaded to nearby edge nodes for processing, thereby improving the QoS of latency-sensitive IoT services. Under the end-edge computing paradigm, the time of downloading service (i.e., dt_{mc}) data under MDS approach is calculated by

$$dt_{mc} = E_{mc} + \frac{wl_{mc}}{c_{mc}} + Q_{mc}, \tag{8}$$

where wl_{mc} is the MDS approach's workload, c_{mc} is the CPU capacity (in MIPS) for the middleware. Q_{mc} is the waiting time in a queue to be executed in the middleware. E_{mc} is the time spent processing data.

Let $\kappa \in [0, 1]$ denote the ratio of locally procedded data, then the overall service delay can be computed as:

$$t_{end-edge} = \max\{\kappa \times et_{local}, dt^{'}_{edge} + (1-\kappa) \times et_{edge}\}, \tag{9}$$

where $dt^{'}_{edge}$ is the newly time of downloading services data at edge node $j \in \mathbb{J}$, i.e.,

$$dt^{'}_{edge} = \kappa \frac{d_v}{Tx_v} + \frac{wl_c}{c_c} + Q_c + \frac{d^{'}_o}{Tx_o}, \tag{10}$$

where $d^{'}_o$ means the output data size of the partially offloaded task size. Then, the explicit form of the overall service delay can be expressed as:

$$t_{end-edge} = \max\left\{\kappa \frac{\alpha_i D_i}{f_i}, \kappa \frac{d_v}{Tx_v} + \frac{wl_c}{c_c} + Q_c + \frac{d^{'}_o}{Tx_o} + (1-\kappa)\frac{\alpha_i D_i}{f_j}\right\}. \tag{11}$$

The energy consumption under the proposed scheme can be computed according to Eqs. (2) and (7).

4.2 Framework of the Proposed MDS Approach

Figure 1 illustrates the detailed workflow of the proposed service provisioning approach with MDS. As shown in Fig. 1, the edge node sends the data generated by the service to the middleware, which is sorted in real time by the MDS component of the middleware, and finally the middleware processes the data in this order.

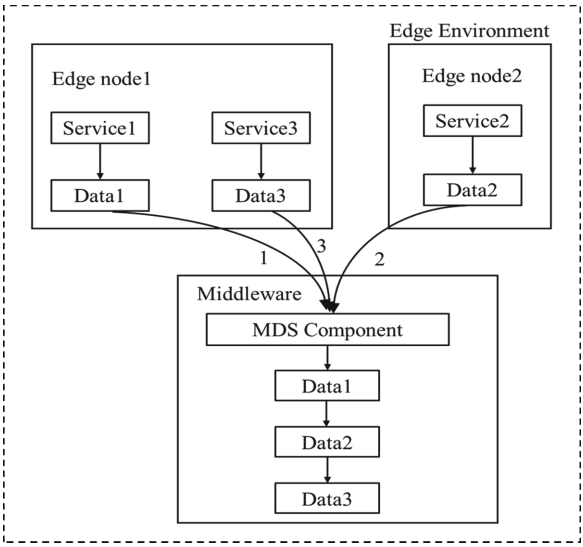

Fig. 1. Framework of the proposed MDS approach.

4.3 Algorithm of the Proposed MDS Approach

The following Algorithm1 is used in the MDS component, which can effectively sort the processing order of the data arriving in the middleware.

Algorithm1

```
int counter = 0;
int winner = randint (0, totaltickets);
job_t *current_job = head;
While (current_job != null)
{
   counter = counter + current_job->ticket;
   if (counter > winner)
      break;
      current_job = current_job -> next;
}
current_job.do ()
```

5 Performance Evaluation

To verify the effectiveness of the proposed approach, two groups of experiments are designed in this paper to compare with the conventional approach based on QT5. This section first describes the experimental settings and then discusses the experimental results.

5.1 Experiment Setting

A hardware embedded with RK3399 is used to conduct the experiment, and its relevant hardware configurations are as follows:

1) CPU performance is not lower than Cotex-AT2;
2) GPU is no less than ARM quad core processor like Mali-T860;
3) Support 2G and DDR3 memory;
4) Support onboard memory which volume not less than 64G like EMMC;
5) Development board gigabit network card;
6) GPIO interface provided by development board.

The detailed software configuration of the experiment is as follows:

project	Software name	Version
Operating system	Debian	Debian9
Library files or supporting software	QT	5.7.0
	GCC	6.3+
	FFmpeg	3.2.9+
programing language	C	C99
	C++	C++ 11
Communication environment	TCP/IP Protocol	SOCKET2.0+
	CAN	CAN2.0B

5.2 Experiment Results

In the first experiment, we verify whether the proposed MDS approach can optimize the processing time when the data generated by multiple services in multiple edge nodes is sent to the middleware. In Experiment Setting 1, the number of edge node services is regarded as a variable, and five edge nodes are built. A total number of 10, 20, and 100 services are distributed on the five edge nodes (average distribution). To ensure that other variables are the same, the same services are used in Experiment Setting 1. Each group of experiments has verified 50 times and taken its average value. The experimental results are shown in Fig. 2.

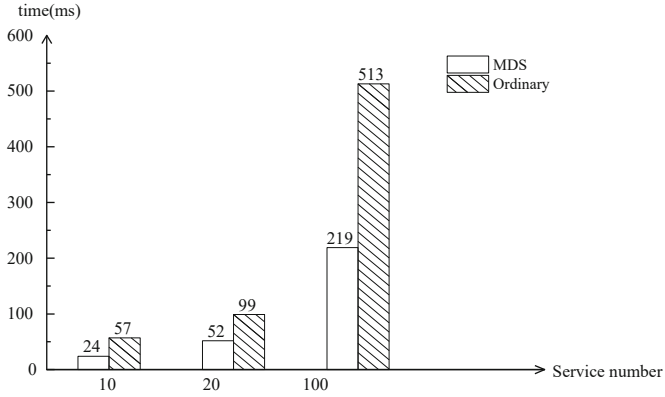

Fig. 2. Evaluation results of the proposed MDS scheme in terms of response time under Experiment Setting 1.

When the number of services is 10, the response time of the conventional approach is 57 ms, and the response time of the MDS approach is 24 ms. If the number of services is increased to 20, the response time of the conventional approach is 99 ms, and the response time of the MDS approach is 52 ms. If the number of services is increased to 100, the response time of the conventional approach is 513 ms, and the response time

of the MDS approach is 219 ms. As seen in Fig. 2, with the increase of the number of services, the proposed MDS approach can always shorten the response time by about 50% compared with the conventional approach.

In the next experiment, four groups of comparative experiments are conducted. In the Experiment Setting 2, 20 edge nodes and 50 edge nodes are adopted to transmit 5M granular data and 50M granular data generated by node services to the middleware respectively. In this way, it is verified whether the MDS approach is still Y when the number of edge nodes and the granularity of data generated by services change. The experimental results are shown in Fig. 3.

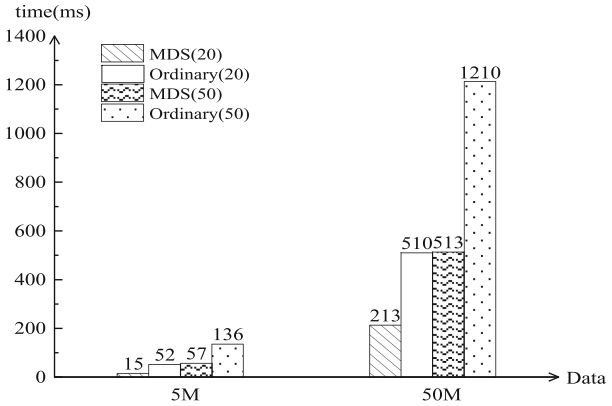

Fig. 3. Evaluation results of the proposed MDS scheme in terms of response time under Experiment Setting 2.

When the data generated by the services in 20 edge nodes is 5M, the proposed MDS approach takes 15 ms to process all the data, and the conventional approach takes 52 ms. When the data generated by the services in the edge nodes are expanded ten times, the proposed MDS approach takes 213 ms to process all the data, and the conventional approach takes 510 ms. Compared with the conventional approach, the proposed MDS approach optimizes nearly 60% of the data processing time. At this time, the number of edge nodes is expanded from 20 to 50, and the experiment is conducted again. When the data generated by the service is 5M, the proposed MDS approach takes 57 ms to process all the data, and the conventional approach takes 136 ms. Nearly 50% of the data processing time is optimized. When the data granularity is increased to 50, the proposed MDS approach takes 513 ms to process all the data, and the conventional approach takes 1210 ms. It can be seen that the proposed scheme can obtain nearly 55% reduction of the data processing time.

6 Conclusion

In this paper, we regard the discovery and scheduling problem of replaceable services replacement as the discovery and scheduling component based on the cache, and develop an optimal approach based on similar replaceable service form the app vendor's perspective for solving the none cached replaceable services environment. Such approach allows adapting of computation from clients, reducing response latency, backbone bandwidth, and the client's computational requirements. We also develop an optimal approach based on similar replaceable service form the app vendor's perspective for solving the none cached replaceable services environment. We have evaluated our approaches on a real-world environment. The results demonstrate that our method can effectively reduce the recovery time after s system failure as the mirror increases.

References

1. Cisco, T., Internet, A.: Cisco: 2020 CISO benchmark report. Comput. Fraud Secur. **2020**(3), 4 (2020)
2. Wang, Y., et al.: A survey on metaverse: fundamentals, security, and privacy. IEEE Commun. Surv. Tutor. (2022). https://doi.org/10.1109/COMST.2022.3202047
3. Lai, P., He, Q., Abdelrazek, M., Chen, F., Hosking, J., Grundy, J., Yang, Y.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: Pahl, C., Vukovic, M., Yin, J., Yu, Qi. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 230–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_15
4. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. **19**(4), 2322–2358 (2017)
5. Wang, Y., et al.: Task offloading for post-disaster rescue in unmanned aerial vehicles networks. IEEE/ACM Trans. Netw. **30**(4), 1525–1539 (2022)
6. Wang, Y., Su, Z., Luan, H.T., Li, R., Zhang, K.: Federated learning with fair incentives and robust aggregation for UAV-aided crowdsensing. IEEE Trans. Netw. Sci. Eng. **9**(5), 3179–3196 (2022)
7. Patel, M., et al.: Mobile edge computing – introductory technical white paper. ETSI White Pap. **11**, 1–36 (2014)
8. Atrey, A., Van Seghbroeck, G., Mora, H., De Turck, F., Volckaert, B.: SpeCH: a scalable framework for data placement of data-intensive services in geo-distributed clouds. J. Netw. Comput. Appl. **142**, 1–14 (2019)
9. Cai, Y., Llorca, J., Tulino, A.M., Molisch, A.F.: Dynamic control of data-intensive services over edge computing networks. arXiv preprint arXiv:2205.14735 (2022)
10. Su, Z., et al.: Secure and efficient federated learning for smart grid with edge-cloud collaboration. IEEE Trans. Industr. Inf. **18**(2), 1333–1344 (2022)
11. Sadeghiram, S., Ma, H., Chen, G.: Composing distributed data-intensive Web services using a flexible memetic algorithm. In: 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, pp. 2832–2839 (2019)
12. Wang, Y., et al.: SPDS: a secure and auditable private data sharing scheme for smart grid based on blockchain. IEEE Trans. Industr. Inf. **17**(11), 7688–7699 (2021)
13. Anantha, D.N., Ramamurthy, B., Bockelman, B., Swanson, D.: Differentiated network services for data-intensive science using application-aware SDN. In: 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Chengdu, China, pp. 1–6 (2017)

14. Cheng, B., Fuerst, J., Solmaz, G., Sanada, T.: Fog function: serverless fog computing for data intensive IoT services. In: 2019 IEEE International Conference on Services Computing (SCC), San Diego, USA, pp. 28–35 (2019)
15. Anisetti, M., Berto, F., Banzi, M.: Orchestration of data-intensive pipeline in 5G-enabled edge continuum. In: 2022 IEEE World Congress on Services (SERVICES), Barcelona, Spain, pp. 2–10. IEEE (2022)
16. Liu, C., Liu, K., Xu, X., Ren, H., Jin, F., Guo, S.: Real-time task offloading for data and computation intensive services in vehicular fog computing environments. In: 2020 16th International Conference on Mobility, Sensing and Networking (MSN), Tokyo, Japan, pp. 360–366 (2020)
17. Chen, Y., Deng, S., Ma, H., Yin, J.: Deploying data-intensive applications with multiple services components on edge. Mob. Netw. Appl. **25**(2), 426–441 (2020)
18. Castro-Orgaz, O., Hager, W.H.: Shallow Water Hydraulics. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-13073-2