Service Migration Based on Replaying

Hexin Zheng[1(✉)], Di Lin[1], Yu Tang[1], Yuan Gao[2], and Jiang Cao[2]

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
201822090541@std.uestc.edu.cn, lindi@uestc.edu.cn
[2] Military Academy of Sciences, Beijing, China

Abstract. With the rapid development of the Internet, more and more data are transmitted on the Internet, and the demand for computing power of end users is also increasing. At the same time, the response delay of the service is becoming more and more sensitive. The traditional data center faces This situation has become increasingly inadequate. Mobile Edge Computing (MEC) has hope to solve this problem. By deploying computing resources to the edge of the network, MEC shortens the distance from users geographically, can handle user requests nearby, improves the speed of service impact, and avoids long network transmissions. A key issue of MEC is to ensure that users always provide services through the geographically closest edge node to ensure service quality. This paper proposes a new idea to migrate user services, by reducing the amount of data transmission in the process of service migration, so as to complete the service migration as quickly as possible.

Keywords: MEC · Service migration · Iterative migration

1 Introduction

With the advent of the "Internet of Everything" era and the popularization of smartphones, the number of various Internet of Things devices and smart terminals has begun to explode, which has also resulted in a substantial increase in network traffic. This makes it increasingly difficult for traditional centralized data centers to meet the network delay requirements of delay-sensitive applications. The proposal of Mobile Edge Computation (MEC) makes it possible to solve this problem [4].

The main idea of MEC is to decentralize computation resources and storage capabilities to the edge of the network to shorten the distance from users geographically. When the terminal sends a request, the request will not go through the long transmission network to reach the data center and then be processed, but locally, directly processed and returned by the MEC server deployed at the edge of the network. Since the request is responded locally, the transmission delay will be greatly reduced, thereby improving the user experience. In addition, data is not transmitted to the cloud data center via the Internet, which also improves data security to a certain extent and reduces the risk of data leakage.

X. Wang et al. (Eds.): AICON 2021, LNICST 396, pp. 537–548, 2021.
https://doi.org/10.1007/978-3-030-90196-7_46

The incoming problem is that because the MEC server is deployed at the edge of the network and its coverage is relatively small, ordinary user movement may cause the terminal device to move from the coverage of one edge node to the coverage of another edge node. In order to ensure the quality of service, the user's service at the previous edge node also needs to follow the user's movement. In addition, in the MEC environment, the network environment between the edge node and the edge node is very complicated, and there may be multiple network topologies and communication systems [22], which makes the bandwidth between nodes extremely limited, which brings a lot of challenges to the rapid realization of service migration.

This paper mainly discusses how to achieve faster user service migration under different bandwidth conditions (Fig. 1).

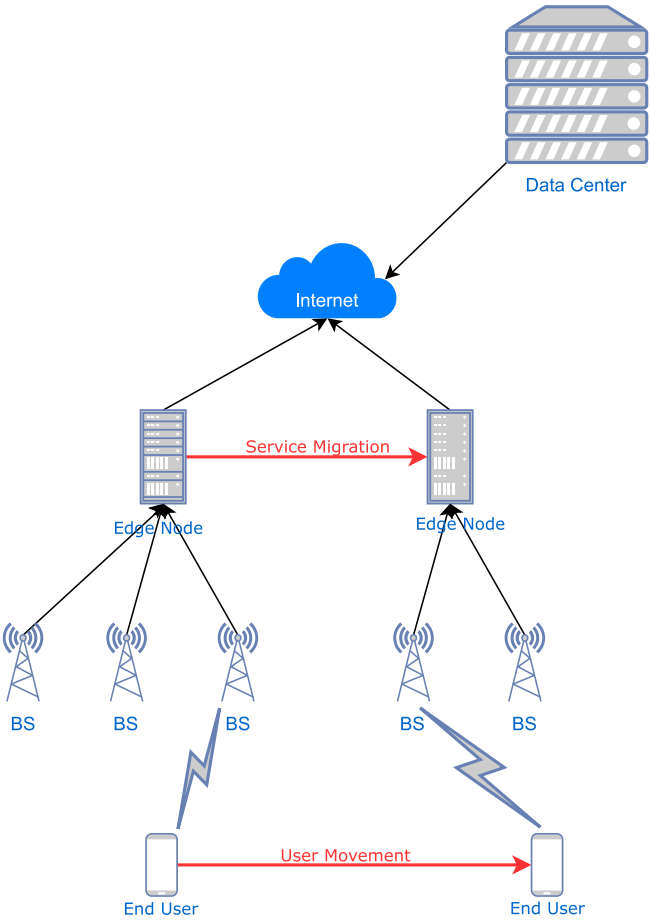


Fig. 1. Service migration in MEC

2 Related Work

Before this thesis, there have been a lot of related researches on user service migration, and the research focus is mainly on the following aspects:

Part of the main research is when and where to migrate. Research in this area mainly uses tools such as Markov Decision Process (MDP) and Reinforcement Learning (RL).

The author of [21] based on MDP, a one-dimensional MDP model based on distance is first proposed. When the user moves under one-dimensional conditions, the moving range is relatively limited and the solution space is relatively small, so it is easier to find the optimal under this condition solution. Subsequently, the author extended the one-dimensional MDP model to the two-dimensional MDP model. In a two-dimensional environment, the user's moving direction is almost infinite, so the author refers to the idea of a cellular network and divides the space into several adjacent regular hexagons. Modeling is then used to simulate the user's movement in space. When solving, the complexity of the direct search algorithm is too high, so heuristic algorithm is used to find the suboptimal solution. [9,10,17,19,20,23] also use MDP in their work about MEC.

Gao in [5] through modeling, abstracts the MEC network model into a graph, each node in the graph represents an edge node, and a node can be connected to multiple nodes. The edges between nodes represent the network between edge nodes. Under the model with several assumptions, the problem can be transformed into a problem that can be processed by reinforcement learning, and then solved by means of reinforcement learning and deep reinforcement learning. Reinforcement learning is also used in the work of others to study service migration strategies in the MEC environment such as [17,18,24,25].

The other part focuses on how to migrate from one edge node to another node faster. Research in this area mainly proposes various solutions to achieve faster service migration. At present, the measurement indicators of the service migration plan are mainly the time required for the entire service migration and the time of the service interruption during the service migration process.

Puliafito mentioned an iterative-based service migration method in [14] (we call it iterative migration). Under the condition of continuous service provision, the data that needs to be transmitted is transmitted to the destination node; when iterating to the threshold, the service is interrupted, and then the remaining data and unsynchronized data are transmitted to the destination node; finally, the service is restored on the destination node to realize the migration. This method shortens the service interruption time caused by service migration through iteration, but this method requires repeated iterations, resulting in a large amount of data transmission, thereby making the entire service migration time-consuming. Kaneda et al. [7] searches the application-level delay in MEC environment by using iterative migration; Kondo in [8] designs a platform which brings cooperative processing between an edge server and a mobile device based on iterative migration. [1,2] also used iterative migration in their work.

Machen proposes a service migration scheme based on layered thinking in [11]. When the traditional solution is used for service migration, related programs will package and transmit many low-level files including the system kernel, resulting in a large number of low-level public files that need to be transmitted. This solution divides the user's services on the virtual machine into three layers: GuestOS, Application, and Instance. When performing service migration, first determine whether the destination node has the same GuestOS as the source node. If they are not the same, the destination node loads the relevant GuestOS. If the same, continue to determine whether there is the binary code of the current service and the dependent files required for service operation in the GuestOS under the destination node. If it does not exist, download the relevant files of the service from the relevant server according to the configuration, and finally transfer the user's service instance. The layered-based service migration idea can greatly reduce the amount of data transmission compared with the traditional solution. Many service migration solutions are based on this idea. The same layering idea is also used in [6,12]'s work

In addition, there are some applied research based on service migration. [16] uses service migration as a tool to maximize the throughput of the regional MEC environment through wireless transmission power control and service migration. The specific idea is that when there is a load imbalance between edge nodes in the area, wireless transmission power control is used to adjust the number of access users of each base station in the area. When the user's access base station changes, service migration is triggered to make the load of edge nodes in this area is balanced, thereby improving the throughput of edge nodes in the whole area.

3 Methodology

In the view of the current migration methods that generally need to transmit a large amount of data, this paper proposes a migration method based on replay (we call it replay migration). This kind of thinking is used in many applications. For example, the database engine InnoDB of MySQL comes with a redo log module, and HBase also uses the Write Ahead Log (WAL) mechanism.

In the current mainstream iterative migration method, when the service migration is triggered, a pre-dump operation will be performed. The file generated by this operation only saves part of the process information and these files cannot be used to restore the service(the file generated by the subsequent dump operation is required to restore the service); then the file generated by the pre-dump operation is transferred to the destination node; after several iterations, the dump operation is performed and the service is suspended, and then the file generated by the dump operation Transmit to the destination node and restore service. Algorithm 1 shows the entire process of iterative migration.

In the replay migration, when the service migration is triggered, it will go through the following stages (assuming the migration from node A to node B):

Algorithm 1: Iterative Migration

```
   // number of iterations can be used instead
   Input: dirt data rate threshold threshold
 1 First pre-dump;
   // ddr means dirt data rate
 2 Calculation ddr ;
 3 while ddr > threshold do
 4 |   Pre-dump ;
 5 |   Transmit data ;
 6 |   Update ddr ;
 7 end
 8 Dump ;
 9 Transmit data ;
10 Restore service at destination node ;
11 Terminate source service ;
12 Start to provide service ;
```

1. Node A generates a snapshot file of the corresponding service, and at the same time redirects the input data of the client to node B; at this time, the client can still provide services
2. When there is data coming in from the client, node B caches the received data and forwards a copy to node A at the same time. When node A returns the calculation result, node B also caches and forwards it to the client; at the same time, node A starts to transmit the generated service snapshot file to node B through the network;
3. After the node snapshot file transfer is completed, node B restore the service according to the snapshot file, and synchronizes the service status according to the cached input data; the service on node A temporarily maintains the status; node B stops receiving data from the client at this time to temporarily stop providing services;
4. When the service synchronization on node B is completed, node A terminates the service; node B stops the forwarding of input data, and starts to provide services to the client;

The whole process of replay migration is shown in Algorithm 2. Figure 2 shows the procedure of replay migration.

Replay migration greatly reduces the amount of data that needs to be transferred by transferring data from the client instead of transferring the memory pages of the service. Due to the small network bandwidth and poor network environment in the MEC environment, reducing the amount of data transmission can effectively speed up the service migration process.

Algorithm 2: Replay Migration

```
1 Dump service ;
2 Cache input data ;
3 Transmit data ;
4 Restore service at destination node ;
5 Suspend source service ;
6 Replay cached data ;
7 Terminate source service ;
8 Start to provide service ;
```

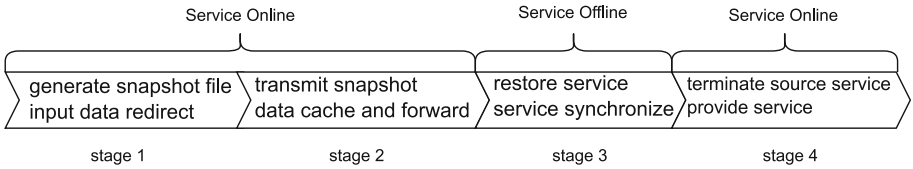


Fig. 2. Replay migration stages

4 Implementation

In the process of method realization, this thesis designed experimental schemes of iterative migration and replay migration respectively, and recorded experiment data for comparison.

In the experiment, we use TensorFlow as the back-end Keras implementation of the YOLOv3 algorithm [15], use the open source image data set Pascal VOC Dataset [13] as the input data, and use the model to identify the pictures in the VOC data set to simulate a service. The two nodes are simulated separately by two computers with the same hardware and in the same LAN environment. The maximum bandwidth between two computers in a local area network is about 580 Mbps. Between the two computers, Wondershaper is used to limit the bandwidth between the two machines. Use iPerf to test the bandwidth between the two machines to ensure that it is within the range we set, file transfer is done by using rsync. The realization of the migration method is mainly realized through CRIU (Checkpoint/Restore In Userspace) [3].

In the experiment, first use Wondershaper to limit the bandwidth between the two computers, and then use iPerf to test and record the available bandwidth between the two computers. Subsequently, CRIU and rsync were used to implement iterative migration and replay migration respectively. After each migration, the relevant logs were analyzed to obtain experimental data.

When implementing iterative migration, we first use CRIU's pre-dump function several times. This operation only dumps part of the information about the process and does this by freezing the task in the shortest possible time. The image file generated by the pre-dump cannot and should not be used for restoration. After each pre-dump is completed, use rsync to transfer the

generated image file. After the transfer is completed, the next pre-dump operation can be performed. After reaching the preset threshold, perform a dump operation and use rsync to transfer related files. After the transmission is completed, use CRIU to restore the service at the destination node. At this point, an iterative migration is completed.

Since CRIU does not have relevant functions to support the completion of replay migration, so replay migration needs to be completed by using some basic functions of CRIU and combining some code changes. When implementing replay migration, first use CRIU's dump operation to generate a snapshot file of the entire service, and keep the service running, at the same time, the operating system sends control information to destination node to start the preset program at the destination node (the program is used to forward and cache related data) and change the destination of input data to destination node; then start to use rsync to transfer the generated snapshot file; when the snapshot file transfer is completed, the operating system will send control information to stop accepting input data and restore the service at the destination node, and then start to calculate the cached data; when the execution of the cached data is completed, it means that the service status has been synchronized and destination node can start to provide service.

In the experiment, by performing iterative migration and replay migration under different bandwidths, the experiment under each condition was repeated three times, and then the average value of the obtained data was calculated as the experimental data. The obtained experimental data are as follows.

From Fig. 3, we can see that when the available bandwidth is reduced to a certain extent, whether it is replay migration or iterative migration, the time required for migration will increase significantly. In comparison, under any bandwidth conditions, the entire service migration time for replay migration is less than 50% of the iterative migration, which fully reflects the advantages of the replay migration method, which is to transmit the input data of the client instead of the memory of the service Pages are used to reduce the time consuming in the transmission process, which in turn consumes the computing performance of the node to ensure that the service can achieve synchronization.

It can be found from Fig. 4 that similar to the migration time, when the bandwidth drops to a certain level, the service interruption time starts to increase significantly. By comparison, the service interruption time of replay migration is much shorter than iterative migration. From the view of migration methods, the service interruption time of iterative migration is mainly concentrated in the last dump operation and the time consumption of subsequent file transfer and service restore, while the service interruption time of recurring migration is concentrated in service restore and replaying cached data. During the migration process, the replay migration only transmits the service snapshot file when the migration is triggered, and because the client's data input has a certain time interval, it can quickly synchronize the service status after the transmission is completed; for the iterative migration, as a result of providing services in the migration's iterative phase, the data in the memory is constantly changing due

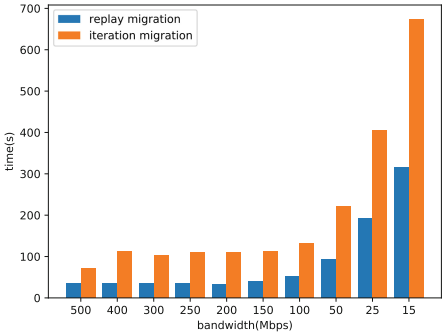


Fig. 3. Total migration time

to processing input data. It is easy to cause the data transmitted in the last iteration to have changed during the next iteration. Due to the existence of this phenomenon, as a result, the file size generated during the last dump operation will increase to a certain extent, resulting in a longer service interruption time for iterative migration.

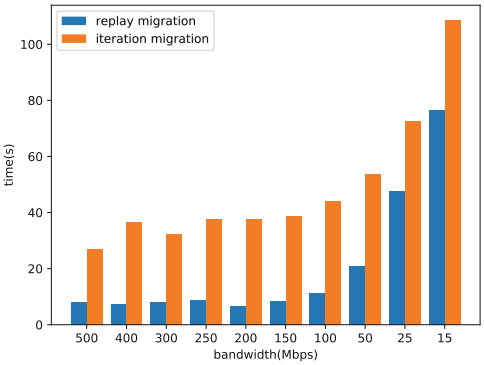
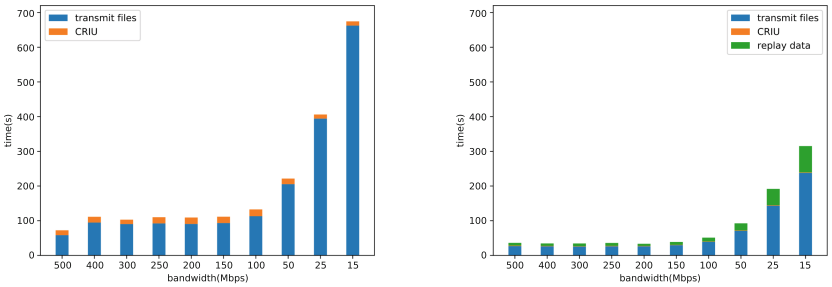


Fig. 4. Interruption time

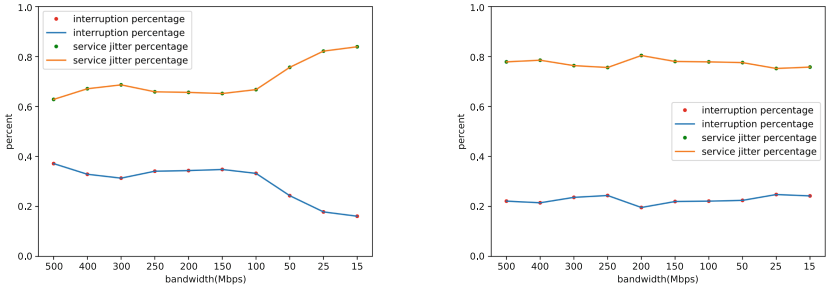
It can be seen from Fig. 5 that file transfer is the most time-consuming part in iterative migration or replay migration. For replay migration, when the bandwidth is reduced, the time it takes to replay the file will increase. The main reason is that the time required to transfer the file increases, which increases the cached data, which in turn requires more time to replay the file. For iterative migration, there are two main reasons why CRIU takes more time than replay migration. On the one hand, CRIU needs to track memory changes, and on the other hand, it reads and processes more files to ensure that the recovered data is up to date when restoring services.



(a) iterative migration (b) replay migration

Fig. 5. Proportion of operation time of service migration

Here we need to introduce a concept called Service Jitter. Service Jitter refers to the fact that the terminal can access the service but cannot access the service through the nearest edge node, which leads to a decrease of the Quality of Service (QoS). Figure 6 shows the ratio of the service interruption duration and service jitter duration of iterative migration and replay migration to the overall service migration duration under different bandwidth conditions. For iterative migration, as the bandwidth continues to decrease, the proportion of service interruption time is generally decreasing. The main reason is that iterative migration requires a large number of files to be transferred under a lower bandwidth, which greatly increases the overall service migration time. Correspondingly, the time proportion of service jitter increases as the bandwidth decreases. For replay migration, the percentage of service interruption time is generally stable and has not changed much with the decrease in bandwidth. In addition, in the case of high bandwidth, the proportion of service interruption time is also lower than iterative migration.



(a) iterative migration (b) replay migration

Fig. 6. Service jitter time and service interruption time

5 Conclusion

In this paper, we first discussed the advantages and some existing problems. Then we introduced the two main research directions in the field of service migration, and introduced in detail the current mainstream iterative migration related content. Then we proposed a recurrence-based migration method, which reduces the amount of data transmission by transmitting user input data instead of memory page files, thereby reducing the time of service migration in a network environment with a small bandwidth. From the experimental results, the replay migration can complete the service migration in a shorter time under suitable broadband conditions, and there is no obvious gap between the service interruption time and the iterative migration.

Acknowledgment. This work is partially funded by Science and Technology Program of Sichuan Province (021YFG0330), partially funded by Grant SCITLAB-0001 of Intelligent Terminal Key Laboratory of SiChuan Province, and partially funded by Fundamental Research Funds for the Central Universities (ZYGX2019J076).

References

1. Addad, R.A., Cadette Dutra, D.L., Bagaa, M., Taleb, T., Flinck, H.: Towards a fast service migration in 5g. In: 2018 IEEE Conference on Standards for Communications and Networking (CSCN), pp. 1–6, October 2018. https://doi.org/10.1109/CSCN.2018.8581836, iM
2. Avramidis, I., Mackay, M., Tso, F.P., Fukai, T., Shinagawa, T.: Live migration on arm-based micro-datacentres. In: 2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC), pp. 1–6, January 2018. https://doi.org/10.1109/CCNC.2018.8319241, iM
3. CRIU. https://criu.org/. Accessed 20 July 2020
4. ETSI: Industry specification group (ISG) on multi-access edge computing (MEC). https://www.etsi.org/committee/mec. Accessed 20 July 2020
5. Gao, Z., Jiao, Q., Xiao, K., Wang, Q., Mo, Z., Yang, Y.: Deep reinforcement learning based service migration strategy for edge computing. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp. 116–1165, April 2019. https://doi.org/10.1109/SOSE.2019.00025, rL
6. Ha, K., et al.: Adaptive VM handoff across cloudlets. School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213 (2015). Layered
7. Kaneda, J., Arakawa, S., Murata, M.: Effects of service function relocation on application-level delay in multi-access edge computing. In: 2018 IEEE 5G World Forum (5GWF), pp. 399–404, July 2018. https://doi.org/10.1109/5GWF.2018.8517045, iM
8. Kondo, T., Isawaki, K., Maeda, K.: Development and evaluation of the MEC platform supporting the edge instance mobility. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 02, pp. 193–198, July 2018. https://doi.org/10.1109/COMPSAC.2018.10228, iM
9. Ksentini, A., Taleb, T., Chen, M.: A Markov decision process-based service migration procedure for follow me cloud. In: 2014 IEEE International Conference on Communications (ICC), pp. 1350–1354, June 2014. https://doi.org/10.1109/ICC.2014.6883509, mDP

10. Lee, J., Kim, J., Tae, Y., Pack, S.: QoS-aware service migration in edge cloud networks. In: 2018 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia), pp. 206–212, June 2018. https://doi.org/10.1109/ICCE-ASIA.2018.8552103, mDP
11. Machen, A., Wang, S., Leung, K.K., Ko, B.J., Salonidis, T.: Live service migration in mobile edge clouds. IEEE Wirel. Commun. **25**(1), 140–147 (2018). https://doi.org/10.1109/MWC.2017.1700011
12. Machen, A., Wang, S., Leung, K.K., Ko, B.J., Salonidis, T.: Migrating running applications across mobile edge clouds: poster. In: Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking, MobiCom 2016, pp. 435–436. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2973750.2985265. https://doi.org/10.1145/2973750.2985265, layered
13. PASCAL VOC TEAM: The pascal VOC project. http://host.robots.ox.ac.uk/pascal/VOC/. Accessed 20 July 2020
14. Puliafito, C., Vallati, C., Mingozzi, E., Merlino, G., Longo, F., Puliafito, A.: Container migration in the fog: a performance evaluation. Sensors **19**, 1488 (2019). https://doi.org/10.3390/s19071488
15. qqwweee. https://github.com/qqwweee/keras-yolo3. Accessed 20 July 2020
16. Rodrigues, T.G., Suto, K., Nishiyama, H., Kato, N., Temma, K.: Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration. IEEE Trans. Comput. **67**(9), 1287–1300 (2018). https://doi.org/10.1109/TC.2018.2818144
17. Tang, Z., Zhou, X., Zhang, F., Jia, W., Zhao, W.: Migration modeling and learning algorithms for containers in fog computing. IEEE Trans. Serv. Comput. **12**(5), 712–725 (2019). https://doi.org/10.1109/TSC.2018.2827070, mDP & RL
18. Vita, F.D., Bruneo, D., Puliafito, A., Nardini, G., Virdis, A., Stea, G.: A deep reinforcement learning approach for data migration in multi-access edge computing. In: 2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K), pp. 1–8, November 2018. https://doi.org/10.23919/ITU-WT.2018.8597889, rL
19. Wang, S., Urgaonkar, R., He, T., Zafer, M., Chan, K., Leung, K.K.: Mobility-induced service migration in mobile micro-clouds. In: 2014 IEEE Military Communications Conference, pp. 835–840, October 2014. https://doi.org/10.1109/MILCOM.2014.145, mDP
20. Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., Leung, K.K.: Dynamic service migration in mobile edge-clouds. In: 2015 IFIP Networking Conference (IFIP Networking), pp. 1–9, May 2015. https://doi.org/10.1109/IFIPNetworking.2015.7145316, mDP
21. Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., Leung, K.K.: Dynamic service migration in mobile edge computing based on Markov decision process. IEEE/ACM Trans. Networking **27**(3), 1272–1288 (2019). https://doi.org/10.1109/TNET.2019.2916577, mDP
22. Wang, S., Xu, J., Zhang, N., Liu, Y.: A survey on service migration in mobile edge computing. IEEE Access **6**, 23511–23528 (2018). https://doi.org/10.1109/ACCESS.2018.2828102
23. Wang, W., Ge, S., Zhou, X.: Location-privacy-aware service migration in mobile edge computing. In: 2020 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6, May 2020. https://doi.org/10.1109/WCNC45663.2020.9120551, mDP

24. Yuan, Q., Li, J., Zhou, H., Lin, T., Luo, G., Shen, X.: A joint service migration and mobility optimization approach for vehicular edge computing. IEEE Trans. Veh. Technol. 1 (2020). https://doi.org/10.1109/TVT.2020.2999617, rL
25. Zhang, M., Huang, H., Rui, L., Hui, G., Wang, Y., Qiu, X.: A service migration method based on dynamic awareness in mobile edge computing. In: NOMS 2020–2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–7, April 2020. https://doi.org/10.1109/NOMS47738.2020.9110389, rL