2019 IEEE 5th International Conference on Computer and Communications

Dynamic Adaptive Network Edge Service Migration Method Based on a Docker Container

Xiangjun Zhang, Weiguo Wu*, Chi Zhang, Wei Song
School of Computer Science and Technology
Xi'an Jiaotong University
Xi'an，Xi'an Shaanxi 710049 P.R. China
e-mail: zxj_2016@stu.xjtu.edu.cn; wgwu@mail.xjtu.edu.cn; chi.zhang;sw578372126}@stu.xjtu.edu.cn

***Abstract*—With the development and maturity of 5G technology, edge networks have become more suitable for moving many low-latency cloud services to the edges of networks, thus significantly improving user service quality. Since mobile terminal users move with their location, the edge computing platform needs to seamlessly migrate services using users' mobile devices. However, the network environment of an edge network is complex, and service migration faces many challenges, such as how to overcome network delays and reduce the service interruption time. In this paper, we propose a service migration method based on Docker base image and design a compression algorithm for an adaptive network environment to compress images and other data. This method balances the multi-core processor computing power and network bandwidth resources. We implement pipeline parallel processing of each process and build a model to handle the situation. The results show that our method is more adaptable to changing network environments, and the migration time is reduced by 45.4%, on average, compared with the traditional uncompressed service switching method.**

Keywords-component; edge computing; service migration; Docker; adaptive compression algorithm

I. INTRODUCTION

In recent years, with the maturity of 5th generation wireless systems(5G) communication technology and the increased variety of sensors, The Internet of Things(IOT) devices such as smart cars and mobile phones can access the Internet through cellular networks, low-power wide-area networks, etc. and can use installed sensors to sense the surrounding states. According to the IDC, the total amount of global data will be greater than 40 zbytes (ZB) by 2020, while 45% of data that are generated by the Internet of Things will be processed at the edges of networks. In this case, the traditional cloud computing model has been unable to meet the high-time computing requirements. When the power of the cloud relies on the device, it results in real-time responsiveness, analysis, and action for applications, especially in areas with limited network conditions . Because edge servers are very close to mobile end users, they can provide a higher quality of service (QoS) than traditional cloud platforms [1]. End users benefit from edge services by offloading heavy computing tasks to nearby edge servers. As a result, an end-user utilizing cloud services will experience higher bandwidth, lower latency, and greater computing power. In this context, many IOT applications have emerged, including real-time data processing in public safety, intelligent networked vehicles and autopilot, VR, the industrial Internet of Things, smart home applications, etc. [2,3].

However, one of the key challenges faced by edge computing [4, 5] is maintaining a better quality of service than traditional cloud services while moving services to the edge servers closest to the end users.when the end user leaves a nearby edge server, the quality of service will significantly decrease due to the deterioration of the network connection. Ideally, when the end user moves, the services on Mobile Edge Computing(MEC) server should also migrate to a closer new server. Therefore, effective dynamic migration [6] is critical to the mobility of edge services in edge computing environments. The method for addressing the edge computing service migration problem is mostly based on virtual machine dynamic migration technology; service migration implements the concept of the "service following the user" [7], which means that the service can move with the user. Although this is a very good method, there are still many challenges in its actual implementation. Because the migration of services is often accompanied by corresponding increases in the network and computational overhead, especially in some practical application scenarios, there may still be a short service interruption. At this point, if the service is frequently migrated, it will result in more negative effects on the edge network, which worsens the edge network and seriously affects users' service experience.

The migration of mobile edge computing services combined with 5G technology has received extensive attention from the industry and academia for several years.[9] Koichi Onoue et al. proposed a method for scheduling multiple virtual machine(VM) parallel migrations with respect to their migration time and migration order. By establishing an integer linear programming (ILP) model to determine the migration order of VMs, researchers have reduced the total migration time in heterogeneous environments and reduced the impact of VM migration on performance degradation.[10] Fei Zhang et al. proposed CBase, which is a VM migration solution for data centers, to effectively support VM migration by introducing a central base image repository to support reliable and efficient data sharing between VMs and data centers. The results were

978-1-7281-4743-7/19/$31.00 ©2019 IEEE

better than those of existing solutions in terms of the total migration time and network traffic.[11] TinYu Wu et al. proposed a memory prediction mechanism for the problem that the VM migration prereplication migration time is too long. According to the characteristics of memory pages, it is possible to determine the optimal time for performing memory migration, thereby reducing the unnecessary migration time and reducing the total migration time. However, the size of the VM overlay above can be tens of megabytes or hundreds of megabytes, which is relatively long for delay-sensitive applications. In addition, virtual machine overlays are not easy to maintain and are limited in industrial and academic applications.

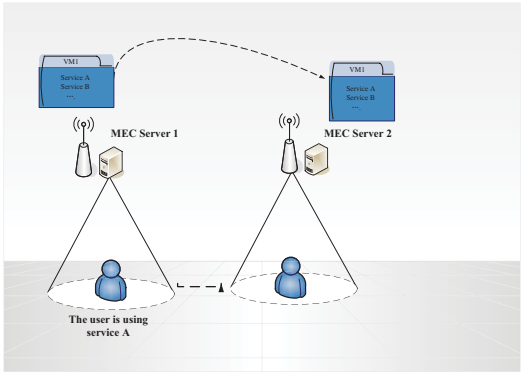


Figure. 1. Scenario of service migration under the MEC

Compared to VMs, the lightweight virtualized Docker platform that is widely deployed in current cloud data centers offers the possibility of service migration at the edges of networks. The Docker container uses the tiered storage file system AUFS to support the fast packaging, sharing, and delivery of any application as a container. The container can be agilely and dynamically implemented; for example, P. Haul [12] developed Docker 1.9.0 and 1.10.0 container dynamic migration, which were developed based on the user-level process checkpoint and recovery tool CRIU [13]. However, this migration method does not perform any processing of the migrated data, and all the data of the storage layer and the entire file system are packaged and transmitted. This method increases the network bandwidth resource overhead.

For the service migration problem, this paper proposes a Docker container-based edge service migration strategy that uses checkpoint technology to generate checkpoints for services and applies compression deduplication algorithms for migrated checkpoint data. Considering the compromise between network bandwidth and computational overhead, the adaptive compression algorithm can dynamically adjust the compression strength to adapt to the network conditions, thereby making full use of the network bandwidth and multi-core processor computing power. The pipeline is processed in parallel to further reduce the migration time. In addition, we establish a mathematical model for the network overhead and computational costs of the migration process and verify the efficiency of our method through comparisons in the VM migration experiment. The main contributions of this paper are as follows.

- To the best of our knowledge, we are the first to apply an adaptive compression algorithm to container-based service migration to minimize the downtime when adapting to the network bandwidth.
- By analyzing the network conditions and dynamically adjusting the compression strength, we achieve a balance between the network overhead and computational overhead. Then, by adopting the shared Docker storage layer technology and pipeline processing for each process, we effectively reduce the calculation and transmission time.
- Our migration method uses a combined view of the data between the source host and target host, and the container can resume operations immediately for the target host without the need for a second data transfer task.

II. EDGE SERVICE MIGRATION

In the cloud computing mode, a user connects to a service through a connection network, which provides a good user experience, and the user and the data connection quality of the data center are closely related. Since the locations of mobile users change at any time, it is difficult for services in the traditional cloud computing mode to make targeted optimizations of the mobility of users. MEC decentralizes computing resources to the edge of the network, which greatly reduces the operating costs of the network and improves the quality of user services. MEC is a promising method that has been well studied. Sensitive, real-time, demanding applications provide a scalable Internet of Things (IOT) architecture [6]. Due to the mobility of users and the limited coverage of edge network servers, network service performance will be degraded, QoS will sharply decrease, and running edge services will be interrupted. Therefore, by addressing these issues, the method improves the continuity of service and provides a good user experience. With respect to the issue of user mobility, the migration of edge services is a promising solution to this problem. The service migration scenario determines when and where mobile users move these edge services after a change in the demand.

A typical migration scenario in an edge network is shown in Fig. 1. First, the mobile user accesses MEC server 1 and obtains service A provided by the server to the user. In addition to service 1, there are many other services on the server. As the user moves, the distance between the user and MEC server 1 gradually increases, and the edge network determines that the service A provided for the user by server 1 should be migrated to the target MEC server 2 to ensure the quality of the service. After the migration is complete, the user no longer needs to continue communicating with MEC server 1 and instead switches to server 2. This processed represents complete service migration, which is transparent to the user.

A. Container Migration

Previous to this article, most of the migration work on containers implemented direct migration between servers

using checkpoint cui technology and did not take into account the impacts of the network environment on the service migration and service interruption time. The adaptive compression algorithm proposed in this paper solves this problem well, and the experimental results proves that the effect is good. Fig. 2 shows the process of a container migration scenario, which is similar to VM migration, except that the migration of containers is more agile and faster.

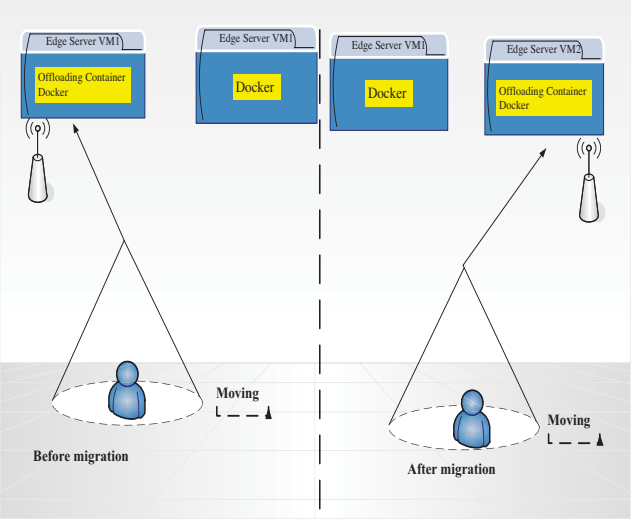


Figure. 2. Container service migration scenario

The Docker container uses checkpoint technology to generate state information for the application. The image of the container is stored in a file system. After unpacking, the file system package becomes another rootfs directory on the

host file system [14]. When the container is instantiated, all of the runtime environments change to the container (for example, new package installations) and any data changes (for example, application states and logs) are conducted by default in rootfs. In addition, any directory in the host file system can be assigned as a data volume to the container. Such a volume can also be a network file system (NFS) installed on the host. Similarly, any block device on the host can also be mapped to the container. However, for volatile states, containers control their memory sharing from the host by using cgroups.

Therefore, migrating containers involves discovering all of the container's container states and configuration-related data and consistently migrating their states (except the memory state) from the source to target host. Memory state migration can be implemented in the user space using the CRIU. In addition, any attached network storage can be migrated by uninstalling it from the source host and assigning it to the target host, which ensures the consistency across all of these states and minimizes application downtime.

B. Migration of Operational Status

Docker uses the CRIU to generate checkpoints for containers, including the many memory states of running containers, such as open file descriptors, network sockets, application data structures, and cached data. We use the open source Linux tool Checkpoint/Restore in Userspace (CRIU) [11] to check the container and dump these states into files that can be migrated and restored. During the checkpoint, the CRIU freezes the running container to maintain its consistency and then dumps its memory state to the directory corresponding to the file.

For checkpoint data storage, we use tmpfs to store the dump file to avoid any further decrease of the disk speed. We further use the CRIU's pageserver model, which directly dumps the pagemap image to the target host during the checkpoint, thus avoiding source host storage hopping. A construct called the action script that allows any script to be executed before the container is selected and unfrozen is also provided in the CRIU. We can provide a callback method in the action script to restore the container state.

III. ADAPTIVE COMPRESSION ALGORITHM

A. Compressed Transmission

Traditional compression algorithms have their own advantages and disadvantages. Directly using these algorithms makes them unable to adapt to network transmissions, which may slow the migration task or result in a poor edge network environment. With respect to low-bandwidth migration in a poor network environment, the methods that are generally used are to slow the running VM or use the post-copy method, leading to unstable application performance, poor user service quality, and delays, which are unacceptable for sensitive applications [15].

Online migration of edge services requires a fast and transparent transfer of running containers from one edge server to another MEC host to ensure the continuity of services. The transmitted data includes status information, such as memory information and disk information. To adapt to the migration of containers in the edge network, we propose an adaptive compression algorithm to make full use of the valuable network resources and dynamically adjust the compression strength according to the network conditions. Edge-based service migration dynamically migrates virtual machines by compressing the migrated data using adaptive compression algorithms. With the advent of multi-core machines or multi-core processors, increasing numbers of available CPU resources have emerged. Even if multiple VMs reside on the same multi-core machine, free CPU resources can still be used because physical CPUs can often be multiplexed. We can use these rich CPU resources to compress the migrated data and make the data more compact, thus effectively reducing the amount of data transferred and retaining more valuable bandwidth resources. In addition, the memory compression algorithm [14] is more efficient for memory page data. Memory compression algorithms usually have very little memory overhead, and decompression does not require memory.

B. Adaptive Network Condition Compression Algorithm

The specific problem solved by the algorithm is how to adjust the strength of the compression algorithm according to the network condition changes so that the transmission data

in the service migration process are more compact and make more efficient use of the network resources. The bandwidth of the edge network is limited, and the amount of data transmitted by the migration task is so large that it increases the service interruption time, which affects the user's service quality. Our algorithm takes into account the computing power of multi-core processors, fully utilizes the network bandwidth resources, and finds the optimal compression time and transmission time. The algorithm is based on the lz4 algorithm. By setting the compression and migration parameters of the compression algorithm to control the compressed stream, lz4 has a total of 9 compression strengths. Level 1 indicates the fastest compression, but the compression ratio is the lowest and the file compression effect is not good. Level 9 indicates that the compression speed is the slowest, but the compression ratio is the highest, making the compressed file much smaller. Our approach is innovative for the following reasons.

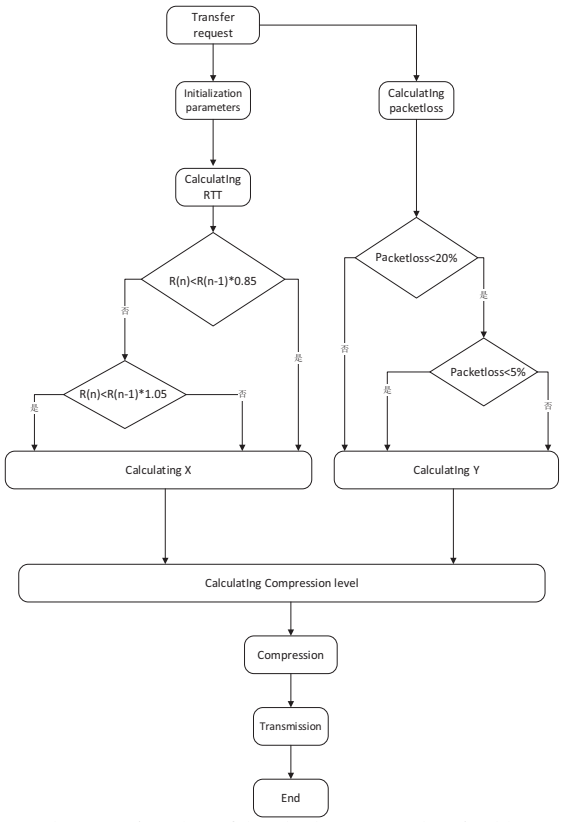


Figure. 3. Flow chart of the adaptive compression algorithm

- In the above compression algorithm, fixed parameter settings cause difficulty when minimizing the service migration time because when using a fixed compression algorithm or compression strength, it is difficult to reduce the compression and transmission time. Therefore, it is necessary to dynamically adjust these parameters as the network environment changes.
- The network bandwidth and the available processing resources can rapidly change over a short period of time. One can continuously monitor the service migration performance and use the tracking information to adjust the processing stage settings and dynamically optimize the migration time. The algorithm's flow chart is shown in Figure 3.

C. *Discrimination of Network Conditions*

The compression strength of the edge service needs to be dynamically adjusted during the migration process. We use the network delay RTT and the packet loss rate packetLoss to assess the current network. The network status function is

$$\text{Netstate} = \text{State}(\text{RTT}, \text{packetLoss}) \tag{1}$$

In Equation 1, RTT stands for the network delay and packetLoss stands for packet loss. RTT sets two thresholds, which are 85% and 105% of the previous RTT. The smaller the ratio, the better the result and the smaller the network delay. PacketLoss also uses two thresholds of 20% and 50%. The smaller the value, the smaller the packet loss rate and the better the network conditions. Based on these thresholds, we can divide the network status into 9 intervals, each of which represents a good or bad current network. Each partition is represented by a location code, as shown in the Figure 4.

Therefore, the space of the network status is {state1, state2, state3, state4, state5, state6, state7, state8, state9}, corresponding to the 9 areas on the chart. For each area's code, the compression level is adjusted each time. The strength of the algorithm will be appropriately increased and decreased based on the compression parameters of the network condition. The default compression strength is 4. The specific adaptation functions are as follows.

$$\Delta_{\text{level}} = Adjust(Netstate) = Adjust(\text{State}(\text{RTT}, \text{packetLoss}))$$

$$\Delta_{\text{level}} = \begin{cases} 4 & \text{RTT} \in [0,0.85] \cap \text{packetLoss} \in [0,0.2] \\ 3 & \text{RTT} \in [0,0.85] \cap \text{packetLoss} \in [0.2,0.5] \\ 2 & \text{RTT} \in [0.85,1.05] \cap \text{packetLoss} \in [0,0.2] \\ 1 & \text{RTT} \in [0.85,1.05] \cap \text{packetLoss} \in [0.2,0.5] \\ 0 & \text{RTT} \in [1.05,\infty] \cap \text{packetLoss} \in [0,0.2] \\ -1 & \text{RTT} \in [0.85,1.05] \cap \text{packetLoss} \in [0.2,0.5] \\ -2 & \text{RTT} \in [0,0.85] \cap \text{packetLoss} \in [0.5,\infty] \\ -3 & \text{RTT} \in [0.85,1.05] \cap \text{packetLoss} \in [0.5,\infty] \\ -4 & \text{RTT} \in [1.05,\infty] \cap \text{packetLoss} \in [0.5,\infty] \end{cases} \tag{2}$$

The representation of 2 shows the increase and decrease of the compression parameters and reflects the status of the network using the network delay RRT and packet loss rate of the current network, thereby obtaining the area. The compression parameters are increased and decreased according to 2. In addition, the expansion parameter of the compression parameter is in the range of [1, 9] and cannot cross the bounds. Thus, we can adapt to the network environment by changing the compression strength of the compression algorithm.

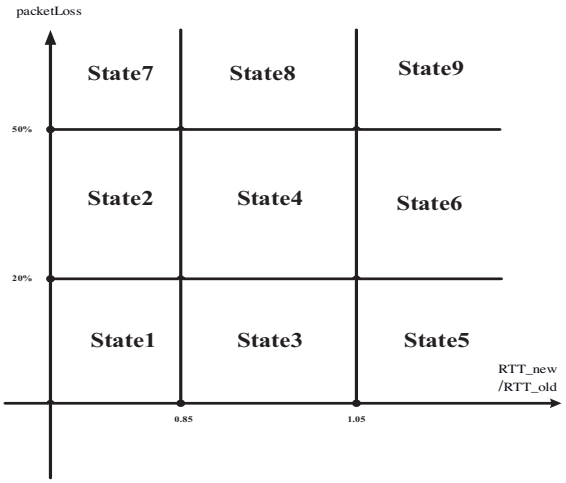
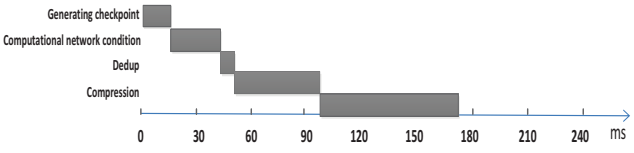


Figure. 4. Network area division

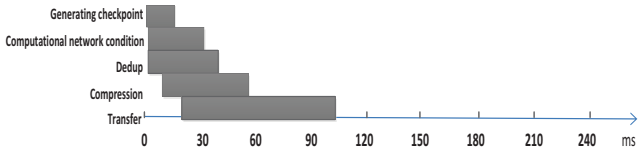
IV. PIPELINE PROCESSING

The data reduction method described in Chapter 3 ensures that the data arrive in the network in a compact manner that reduces bandwidth usage and fills the network as quickly as possible. However, due to the CPU processing intensive processes because of the generation of checkpoints for the compressed transport of the migrated data [17], serial execution may cause the network to experience significant delays before receiving any data to be transmitted, and the network is idle during these delays. These delays will result in a lost opportunity to utilize network resources. Keeping the network full of valuable data is critical to improving the performance in low bandwidth conditions [18]. As mentioned above, some processes that are not correlated can be processed simultaneously, which can result in two benefits to the pipeline.

- The downstream phase can be started before the previous phase is completed. For example, the checkpoint generation process and the quality of the network according to the conditions can be used to determine the change of the compression level in advance, and the compression, transmission, and calculation processes can be processed in parallel.



(a) Serial processing time



(b) Pipeline processing time

Figure. 5. Serial and parallel processing comparison

- Caching the temporary data generated by a single phase requires less memory, which reduces the memory required to buffer the intermediate data generated in the upstream phase because they are quickly removed by the downstream phase.

The results show in Figure 5 show that pipelined parallel processing multiple processes saves nearly 46.1% of time compared to the serial execution migration time.

V. COMPRESSED TRANSMISSION PROBLEM FORMALIZATION

A. System Throughput

To quantify the impact of the pipelined parallel processing of multiple processes on the throughput of the system, we build a mathematical model of the pipeline model.

We use P_i to denote the time consumption of the i-th pipelined process, R_i to denote the compression ratio, and $Size^{i}_{input}$, $Size^{i}_{output}$, and $Size^{i}_{migration}$ to denote the input data, output data, and size of the migration data of the i-th process, respectively. Then, we formulate the following:

$$P_i = processing\ time \qquad R_i = \frac{Size^{i}_{output}}{Size^{i}_{iuput}}$$

Therefore, the processing time and transmission time are, respectively, denoted as follows:

$$Time_{processing} = \sum_{1}^{n} P_i \tag{3}$$

$$Time_{transfer} = \frac{Size^{i}_{migration}}{Network\ \ bandwidth} \tag{4}$$

The processing throughput and transmission throughput of the system are as follows:

$$TPS_{processing} = \frac{1}{\sum_{1}^{n} P_i} \tag{5}$$

$$TPS_{transfer} = \frac{\text{Network bandwidth}}{\prod_{1}^{n} R_i} \tag{6}$$

Migration time and system throughput are as follows:

$$Time_{migration} = \max(Time_{processing}, Time_{network}) \tag{7}$$

$$\mathrm{TPS}_{\mathrm{system}} = \min(TPS_{\mathrm{processing}}, TPS_{network}) \quad (8)$$

Intuitively, (8) and (6) indicate that the system throughput and migration throughput can be estimated using the processing time (P), the current network bandwidth, and the access rate (R). It can be seen that P and R of the network are critical. According to the throughput of the above pipeline, the migration mode can be selected to maximize system throughput [19]. We use P={ p_i |i=1~n} and R={ R_i |i=1~n} to denote the various parameter settings. However, they depend to a large extent on the actual transmitted data, and thus, P and R can provide highly misleading results. We find that the trends of P and R are similar in different scenarios and that the ratio R under different workloads is also significantly different. Although one workload [20] may be quite different from another, its impact on different algorithms is similar and the relative performance remains the same. The compression algorithm that we use dynamically adapts to changes in the network bandwidth, dynamically adjusts the compression strength, and uses a memory compression algorithm to compress the memory pages that account for a large proportion of the migrated data. It can adapt to changes in the network bandwidth and make full use of the computing power of a multi-core CPU and the network bandwidth.

B. Transfer Model

Usually, the compression algorithm has its own compression ratio and compression time. The lz4 compression algorithm that we use is the same. The higher the compression ratio of the compression algorithm, the longer the compression time. If only the compression ratio is pursued, the increase in the compression time will make the total migration time longer. If the compression speed is pursued, the data that may be transmitted are not compact. The time will be longer, which is not conducive to a shorter migration time. We establish the following model to demonstrate the compression migration. Here, the compression ratio of the compression algorithm is C_{ratio}, the compression speed is $V_{compress}$, and the data with the data size of D_{size} is compressed and transmitted.We defined $T_{compressing}$ as compression time, $T_{transfer}$ as transmission time, $T_{\text{non-compression}}$ as migration time without compression, and $V_{network}$ as network bandwidth , the following can be obtained.

$$T_{compressing} + T_{transfer} \le T_{\text{non-compression}} \quad (9)$$

$$\frac{D_{\text{size}}}{V_{\text{compress}}} + \frac{D_{\text{size}} C_{ratio}}{V_{network}} \le \frac{D_{\text{size}}}{V_{network}} \quad (10)$$

The above formulas can be deformed to obtain the following.

$$V_{\text{compress}}(1 - C_{ratio}) \ge V_{network} \quad (11)$$

It can be seen from Equation 11 that both the compression speed $V_{compress}$ and the compression ratio C_{ratio} are key to satisfying the above formula. The faster the compression speed $V_{compress}$, the larger the left multiplier factor, and the higher the compression ratio, the smaller the value of C_{ratio} . The larger the value of (1- C_{ratio}), the larger the multiplier factor on the right.

However, in fact, the compression algorithm that achieves both a fast compression speed $V_{compress}$ and a high compression ratio C_{ratio} is relatively small. Generally speaking, the compression ratio C_{ratio} is inversely proportional to the compression speed $V_{compress}$, and an algorithm with a high compression ratio tends to have a slower compression speed. Therefore, a compression algorithm that is applicable to compressed transmissions in real-time edge service migration should have a moderate compression ratio C_{ratio} and compression speed $V_{compress}$.

In summary, finding a compression algorithm that is suitable for compressed transmissions of the real-time migration of containers is key to the research of migration methods. In fact, in the Docker container running the cui to generate checkpoint data, the total size that needs to be dumped is mainly determined by the size of the pages-*.img and pagemap.img. Because the memory page data in the compressed checkpoint data are relatively large, based on the memory page characteristics, we use the WKDM [16] memory compression algorithm to process the compressed memory page. Other data volumes and metadata can be dynamically set to compress the lz4 compression algorithm. Wkdm is a recent time based compressor that uses machine learning, direct mapping, a 16-word dictionary and fast coding. The WKDM algorithm achieves the best performance for most data points due to its speed and compression ratio being comparable to LZO.

VI. EXPERIMENTAL ANALYSIS

A. Experiment Descriptions

we use three blade machines. One is the master node, which is responsible for distributing the migration tasks, and the other two are the source servers and migration target servers. Each of the blades runs a different Docker service and passes from one blade to another and recovers from the target. We use Wondershaper to adjust the different bandwidths for testing. The simulation environment is as follows: the Ubuntu 16.04 operating system for each edge, 2 physical CPUs, 24 logical CPUs, a 2.4 GHz Intel Xeon E5645 with 6 cores, and 32GB of memory. The hard disk is a Toshiba MBF2300RC-0109 (4TB).

In this section, we test the total migration time and throughput in the real-time migration of different applications. In this test, to avoid the loss of generality, three applications are selected in this experiment: apache2, MySQL, and tomcat. Among them, apache and tomcat7 are network-intensive applications[21].In this experiment,apache is used as a static web server. The version number is apache-2.4.18, and tomcat is used as a dynamic web server. The version number is apache- Tomcat-7.0.82, and the Linux

kernel version is 4.4.0. The test results were obtained by using multiple tests and averaging the results.

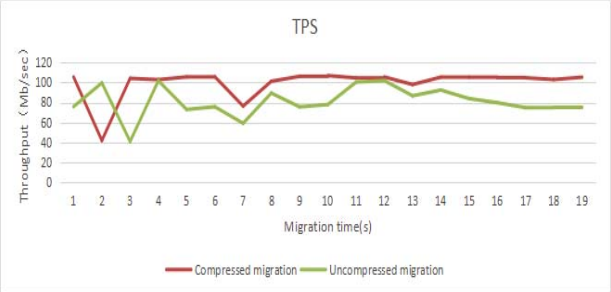


Figure 6. Compressed and uncompressed method throughput comparison

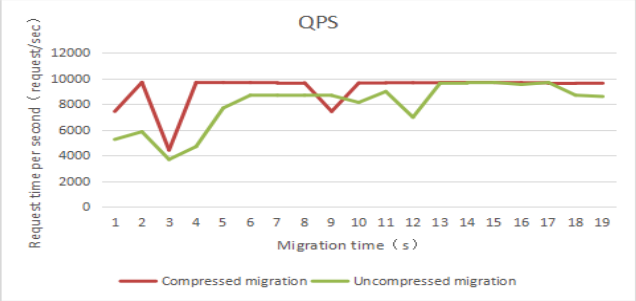


Figure 7. Comparison of the query rates per second

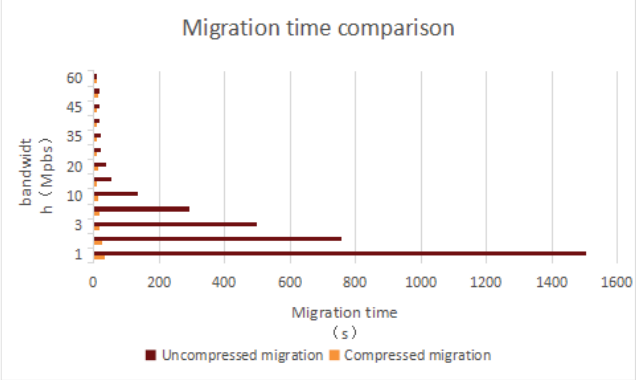


Figure 8. Comparison of the migration times of the compressed and uncompressed methods

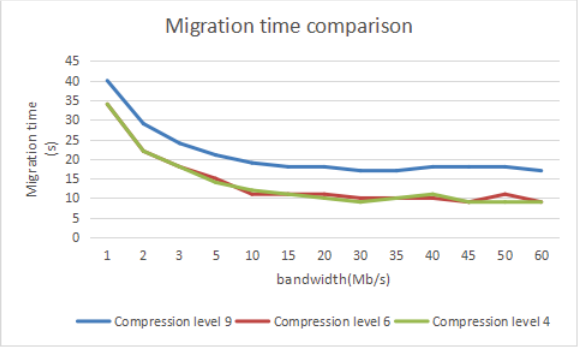


Figure 9. Comparison of the migration times of the different compression levels

B. *Migration Time Comparison*

Figure.8 provides a comparison of the compression time between the compression algorithm and the non-compression algorithm. The results show that the migration time of the compression algorithm is significantly better than that of the non-compression algorithm. Figure 9 compares the migration times at different compression intensities. It can be found that different compressive strengths have limited effects on migration. When the computing power is sufficient, the compressive strength is up to 9, and the migration time will increase because the compression takes
too long. When the situation is the opposite, the compression is rapid and the migration is slow. Faster compression results in less migration time, which also confirms the theory presented in Section 5 that the compression strength needs to be adapted to the network environment to achieve the minimum migration time.

C. *Throughput Rate and Query Rate Per Second*

The above compares the migration time. Next, the throughput (TPS) of the system during migration and the query rate per second (QPS) are analyzed. The QPS is a measure of how much traffic is processed by a particular query server within a specified time. It is used to reflect the performance characteristics of the external migration service that is provided by the application migration process. The test uses apache for testing. The apache service is started in the Docker container. A test ab (apache benchmark) is started on a client. The page that is accessed by the ab is a static page that is 512Kb, and the number of concurrent calls is 100. To test the results as accurately as possible, we set the number of AB requests to 50,000. The command is as follows:

"ab -c 100 -n 50000 http://daopian1:4004/"

The QPS and TPS of the system during the migration process can be derived from the throughput per second and the request field of the multiple return instructions. The ab command is used to test the compression-based service migration method and the original non-compression migration method. The test results are shown in Figure 6. From the throughput rate, we can see that the throughput of the compression-based service migration method fluctuates between 40.4 Mb/s and 107.6 Mb/s during the entire migration process, and the average throughput rate is 74 Mb/s. From left to right, the total migration time is relatively short at approximately 13 seconds. The original non-compressed migration method has a longer migration time, and the migration time is approximately 18 seconds. Especially at the 2nd to 14th seconds of the migration, the throughput of the adaptive compression algorithm is higher than that of the non-compressed migration service. Figure 7 shows the comparison of the query rates per second (QPSs) of the two methods. It can be seen that the compression migration method based on the Docker container can still maintain a good external response during the migration. The overall response requests per second are high in the traditional non-compressed migration method.

VII. CONCLUSION

Edge computing applications need to evaluate mobile user terminals' location movements and make a decision of whether to migrate the task to an edge server closer to the user. The traditional VM migration method is cumbersome and imposes burdens on the edge network bandwidth, and its migration is inconvenient. Our proposed Docker container-based migration strategy uses the CRIU checkpoint technology of the Docker container to store the container state by sharing the base image in the edge network and then adopts an adaptive network state compression algorithm for the migrated data. The algorithm can dynamically adapt the compression strength after evaluating the conditions of the edge network, and it comprehensively balances the computational overhead and network transmission overhead. In addition, we introduce pipeline parallel processing multitasking that fully uses the computing power of multi-core processors. The experimental results show that compared with the traditional non-compressed service migration method, our method improves the adaptability of service migrations in complex network environments, and the migration time is greatly improved. The system can provide stable responses during migration. In the case of migration experiments with different bandwidths, our pipelined parallel processing multitasking combined with the adaptive compression algorithm migration method is more efficient than the traditional non-compressed migration method. The average migration time savings is close to 45.4%.

ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Program of China under Grant No.2017YFB0203003,the National Natural Science Foundation of China under Grant No. 91630206 and 61672423.

REFERENCES

[1] Rec. ITU–R M.2083-0, "IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond," Sep. 2015.

[2] Shi Weisong, Zhang Xingzhou, Wang Yifan, Zhang Qingyang. Edge Computing: State-of-the-Art and Future Directions. Journal of Computer Research and Development, 2019, 56(1): 69-89.

[3] Weisong S , Hui S , Jie C , et al. Edge Computing——An Emerging Computing Model for the Internet of Everything Era[J]. Journal of Computer Research & Development, 2017.

[4] Ahmed E , Rehmani M H . Mobile Edge Computing: Opportunities, solutions, and challenges[J]. Future Generation Computer Systems, 2016, 70.

[5] Gahlot L K , Khurana P , Hasija Y . A Smart Vision with the 5G Era and Big Data—Next Edge in Connecting World[M]// Advances in Mobile Cloud Computing and Big Data in the 5G Era. Springer International Publishing, 2017.

[6] Shi W , Fellow, IEEE, et al. Edge Computing: Vision and Challenges[J]. IEEE Internet of Things Journal, 2016, 3(5):637-646.

[7] Wei X , Wang S , Zhou A , et al. MVR: An Architecture for Computation Offloading in Mobile Edge Computing[C]// 2017 IEEE International Conference on Edge Computing (EDGE). IEEE, 2017.

[8] Upadhyay A , Lakkadwala P . Secure live migration of VM's in Cloud Computing: A survey[C]// 2014 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions). IEEE, 2014.

[9] Onoue K , Imai S , Matsuoka N . Scheduling of Parallel Migration for Multiple Virtual Machines[C]// IEEE International Conference on Advanced Information Networking and Applications. IEEE, 2017.

[10] Zhang F , Fu X , Yahyapour R . CBase: A New Paradigm for Fast Virtual Machine Migration across Data Centers[C]// IEEE/ACM International Symposium on Cluster. IEEE Press, 2017.

[11] Wu T Y , Guizani N , Huang J S . Live migration improvements by related dirty memory prediction in cloud computing[J]. Journal of Network & Computer Applications, 2017, 90:83-89.

[12] P.Emelyanov.(2017)Live migration using criu. [Online]. Available: https://github.com/xemul/p.haul

[13] CRIU. (2017) Criu. [Online]. Available: https://criu.org/Main_Page

[14] Nadgowda S , Suneja S , Bila N , et al. Voyager: Complete Container State Migration[C]// IEEE International Conference on Distributed Computing Systems. IEEE, 2017.

[15] Jin H , Deng L , Wu S , et al. MECOM: Live migration of virtual machines by adaptively compressing memory pages[J]. Future Generation Computer Systems, 2014, 38:23-35.

[16] Wilson P R , Kaplan S F , Smaragdakis Y . The Case for Compressed Caching in Virtual Memory Systems[C]// Usenix Technical Conference. 1999.

[17] Ha K , Abe Y , Eiszler T , et al. You can teach elephants to dance: agile VM handoff for edge computing[C]// Acm/ieee Symposium. ACM, 2017.

[18] Satyanarayanan, Mahadev. The Emergence of Edge Computing[J]. Computer, 2017, 50(1):30-39.

[19] Qiu Y , Lung C H , Ajila S , et al. LXC Container Migration in Cloudlets under Multipath TCP[C]// Computer Software & Applications Conference. IEEE, 2017.

[20] Sun X , Ansari N . EdgeIoT: Mobile Edge Computing for the Internet of Things[J]. IEEE Communications Magazine, 2016, 54(12):22-29.

[21] Samie F , Tsoutsouras V , Bauer L , et al. Computation offloading and resource allocation for low-power IoT edge devices[C]// Internet of Things. IEEE, 2017.