International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-6, March 2020

Live Migration of Stateful Processes across Edge Servers

Harkiranpreet Kaur, Kiranbir Kaur

Abstract*: Edge Computing facilitates low-latency for real-time applications. It can be achieved by installing computing infrastructure at the network edge or bringing cloud benefits closer to the user. In Edge Computing, computing resources are not more than one hop away from the user equipment. The mobility of a user in such a case can cause problems. If a user continues to move, maintaining low-latency becomes challenging. So, service is migrated between edge servers as the user moves. This migration can lead to a time duration for which the service is not available, called Downtime. While migrating, to leverage low- latency, Downtime must be kept to a minimum. In this paper, we introduced an approach to migrate service so that the minimum amount of data is transferred during Downtime. We have also discussed some existing techniques for migrating service and compared them based on how they reduce data to be transferred to lessen the Downtime. Our approach transfers lesser data that causes less Downtime than one of the existing techniques.

***Keywords*: Edge Computing, Live Migration, Low-Latency, Process Migration**

I. INTRODUCTION

Many real-time applications are increasing day by day. These applications require computing resources in the proximity of equipment. The sensors, actuators, drone-related applications, intelligent transportation systems (ITS) [1], being used these days require computational results in the minimum time. Offloading intensive computations to the cloud [2] can be the best solution to achieve this, but it has its problems. The geographical distance between the cloud and user equipment causes a delay in results. Offloading to cloud solves the problem of lacking resources but gives a high latency.To handle this problem, we need to install computational resources closer to the user equipment. This approach is called Edge Computing [3] i.e., Bringing resources at the edge of the network. This solution can also cause a problem if there is no support for the mobility of the user [4]. The mobility of a user requires mobility of service. Service is migrated between different hops as the user continues to move. The frequency of migrations affects the latency observed by the user.

Manuscript received on March 15, 2020.
Revised Manuscript received on March 24, 2020.
Manuscript published on March 30, 2020.
* Correspondence Author
Harkiranpreet Kaur*, M. Tech. degree, Guru Nanak Dev University, Amritsar.
Kiranbir Kaur, Assistant Professor, Computer Engineering and Technology department of Guru Nanak Dev University, Amritsar.

Downtime during the migration is the primary reason behind this increased latency. The more the Downtime, the more it affects applications that require ultra-low latency. In Section 2, we have discussed some techniques that help in reducing Downtime for these types of applications.

Need for mobility support- As shown in Figure 1, suppose a mobile user has offloaded its application to the edge server because computational resources lack in mobile devices. Being directly connected to MEC1, a user is seamlessly receiving service with low-latency. As the user starts to move, the distance between a user and MEC1 continues to increase. Itresults in high latency [5]. To get benefit from edge computing, we transfer the service to MEC2 so that users can get service again without any delay.

II. EXISTING TECHNIQUES

A. Two-Layer approach

The Two-Layer approach migrates service from source Mobile Edge Cloud (MEC) to destination MEC. As the name suggests, the transfer of files between MECs uses two layers, say, Base Layer, and Instance Layer. This approach works for Virtual machines as well as Containers [6].

Base Layer consists of a package that contains all the data related to OS, Kernel, virtualization, whereas the Instance Layer contains applications installed and their running states. The size of the package of the Base Layer is about a few MBs for containers and a few GBs for Virtual machines. The migration of these packages takes a few seconds during which service is down. To reduce the Downtime, make Base Layer generic for all Mobile Edge Clouds, and every server has a copy of it [6].

Assuming Base Layer is already present on the destination server, whenever required; transfer only the Instance Layer from source to destination MEC. The service on source MEC stops, and then transfer occurs. The service resumes after combining the Instance Layer and already present Base Layer. As we only require transferring the Application Layer, the amount of data transfer reduces, which in turn reduces the service Downtime [6].

B. Three-Layer approach

Even though the amount of data required to transfer got reduced in the Two-Layer approach, we still cannot achieve short Downtime of service. For better performance, the Three- Layer approach [6] to transfer data between two Mobile Edge Clouds got proposed.

The new layer is an intermediate Application Layer, containing an idle version of the application. Instance Layer now contains only the running state of the application. Base Layer remains the same.

Similar to the Two-Layer approach, Base Layer is already present on every MEC. When the migration process starts, the service on source MEC keeps running. The Application Layer is transferred to the destination MEC. Then the service stops on the source MEC, and Instance Layer migrates to the destination. This approach reduces the Downtime as compared to the Two-Layer approach because Downtime is present only while migrating the Instance Layer.

The performance of this approach is studied using KVM and LXC [7].

C. Three-Layer approach with Iteration

Improvement has done in a Three-Layer approach by in- troducing iteration in the migration of the Instance Layer. In the Three-Layer approach, Downtime starts with the migration of the Instance Layer. In this approach, the migration of the Instance Layer uses Pre-Copy Migration [8]. The running state of an application migrates using 'n' iterations. In the first iteration, transfer the complete copy of the in-memory state. In the next 'n-1' iteration, transfer only dirty pages until the rate of data getting transferred is less than the page dirtying rate. Then stop the service on source MEC and transfer data of the last iteration to destination MEC. In this approach, Downtime is the time taken by the last iteration only.

D. Service Handoff across Edge Servers via Docker

This approach (refer [9]) proposes the use of layered storage system, selecting a target edge server, the Base Layer is synchronized for the Docker container. Then container run time memory is dumped and transferred to the target edge server. From now, service handoff starts and halts service on source edge servers, and the latest run time states get transferred to target edge servers. After checkpointing, run-time states remain the same, and both source and edge servers are synchronized. The next step is to reload Docker daemon to make target container daemon able to recognize configuration files created by the source edge server. Then compare the final dump to pre-dump memory and transfer only those files, which are not present on the target server. The target server restores the container.

E. Stateful Process Migration for Edge Computing Applica- tions

To further reduce Downtime, this method proposes to trans- fer only Stateful Processes rather than an application and its in- memory states [10]. A server-side application contains Stateful and Stateless Processes. Stateful Processes contain user equipment related states, while Stateless Processes only provide services to Stateful Processes. Stateful Processes are reconnected to the Stateless Processes (already running on the destination server) to continue service. This method reduces the amount

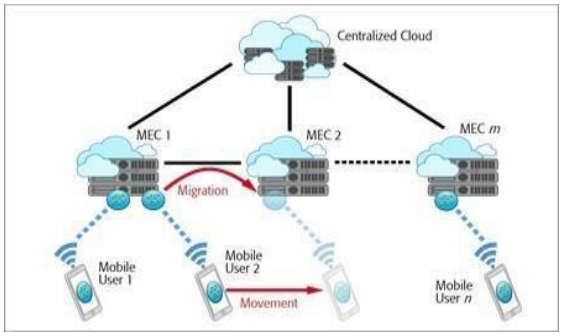

Fig. 1. [6] Motivation behind Migration

of data to send over the network, which results in a decrease in Downtime [10].

This method consists of three phases:

Preparation - Before the process of migration begins, we have to make sure that OS and stateless processes are already running on the destination server.

Migration - In this step, the migration of Stateful Processes occurs. Processes are migrated to the destination server only if resources are available; otherwise, migration fails.

Reconnection - For establishing reconnection between Stateful and Stateless Processes- Interprocess Communication Channels (IPC) are converted i.e. Request for communication gets convert to request for a newly created communication channel.

III. COMPARISON OF EXISTING TECHNIQUES

Table I shows the type of data that above discussed techniques target to migrate after the service stops running on the source-side. Base Layer contains the OS, which is generic for all servers, and is not usually migrated. Application Layer contains the idle version of Application [6] which has both Stateful and Stateless processes. Instance Layer contains the in-memory state of running application.

IV. PROPOSED METHODOLOGY

Earlier, a novel approach was introduced to transfer only Stateful Processes to the destination server instead of trans- ferring the Virtual machine, containers, or a whole applica- tion [10]. These Stateful Processes are connected to Stateless Processes. The Stateless Processes are transferred to the destination servers before the Stateful Process is migrated to minimize the Downtime.

The method contains three steps. In the first step, the oper- ating system and the Stateless Processes are transferred to the destination machine. When the migration triggers, the second step starts, which involves the transfer of Stateful Processes from source to the destination server. It stops application on the source server and transfers Stateful Processes to the destination server. Communication channels are created between Stateful and Stateless Processes on the destination in the third step. This approach reduces the amount of data to be migrated, which results in reducing the Downtime of the application.

We can further reduce this data that needs to be migrated on the destination server by introducing Live Migration in the second step of the approach mentioned above.
There are two ways to perform migration –

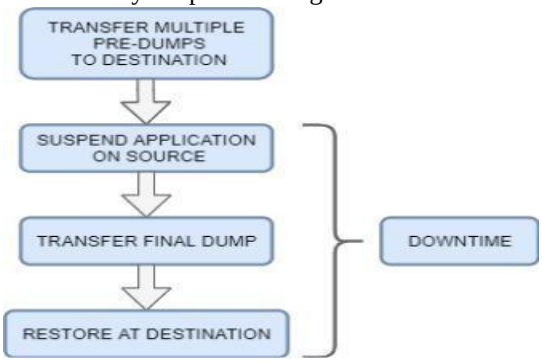

Fig. 2. Downtime caused by Live Migration

Simple Migration- This is the existing method. The whole application stops, and the process state is transferred to the destination server, and then the process is restored on the des- tination server using this process state. This method involves transferring the data for the application to the destination server after the service is suspended on the source server. This approach doesn't scale well with the size of the process. To overcome this deficiency, we propose the following updates to the migration process.

Live Migration- Live Migration differs from Simple Migration in migrating some of the process state before stopping the application on the source server. We transfer a part of the process states prior to stopping the process with the aim that when we stop the application on the source server, we are left with a smaller amount of process state to transfer to the destination server. The process includes the determination of when to stop the process at the source server and transfer the remaining process state to the destination server.

TABLE I
TECHNIQUES ARE COMPARED BASED ON THEIR MIGRATION TARGETS

Techniques	Migration includes Base Layer	Migration includes Application Layer	Migration includes Instance Layer	Migration includes Stateless Processes	Migration includes Stateful Processes
Service Migration using Two-Layer Approach [6]	No	Yes	Yes	Yes	Yes
Service Migration using Three-Layer Approach [6]	No	No	Yes	Yes	Yes
Service Migration using Three-Layer Approach with Iterations [8]	No	No	Yes	Yes	Yes
Service Handoff across Edge Servers via Docker [9]	No	Yes	Yes	Yes	Yes
Stateful Process Migration for Edge Computing Applications [10]	No	No	No	No	Yes

Following steps are followed for Live Migration in our methodology-

- Once the intention to migrate a Stateful Process is made clear, we start transferring the process state to the desti- nation server without stopping the process on the source server.
- Each successive process state capture includes only those pages which have changed since the previous process state capture. Once no such change is detected between two successive process states, we make a reasonable assumption that the process is idle during this time, and we can stop this process at the source server.
- The remaining state is migrated to the destination server after stopping the application on the source server.

Of course, the criteria to decide when to stop and migrate the process can be changed based on different application needs. In this paper, we present the result using the criteria described above.

V. DOWNTIME CALCULATION

In Stateful Process Migration, by applying Live Migration, better Downtime can be achieved.

Procedure-

1) Take Process ID, Source Path, Destination name, Desti- nation IP, Destination Path as input to Algorithm.
2) Save the process state (let's call it the snapshot) for the first time and transfer it to the destination server (do not stop the process at this point).
3) Save the process state that has changed since the last snapshot of the process state (let's call it the incremental snapshot) without stopping the process and transfer to the destination server.
4) Calculate hash for the last two incremental snapshots of the process state.
5) Repeat steps 3 and 4 until the hashes taken in step 4 are equal. This indicates that there was no change in the process state between the last two incremental snapshots.
6) Take a final incremental snapshot of the process state and stop the process on the source server (Downtime Starts).

7) Transfer process state to the destination.
8) Restore the process on the destination server using the transferred process state *(Downtime Ends).*

Figure 2 shows the different steps involved and Downtime caused by Live Migration.

VI. EVALUATION ENVIRONMENT

For Live Migration, we used CRIU [11] (Checkpoint/ restore in Userspace), a software tool for Linux. It provides support for migration [12]. Iterative Migration in CRIU [13] helps in achieving Live Migration as follows-

Pre-dump: Each pre-dump command in CRIU captures the process state relative to the process state captured in the last pre-dump command, i.e., this command captures the changes to the process state [13]. Until the captured process state meets the criteria defined above to stop the process, continue taking these pre-dumps. The time taken in this step is not counted towards the Downtime because we do not stop the process during this step.

Dump: It dumps (stops) the process [13]. *Dump time* is the time taken by this phase.

Copy: It involves transferring data between two servers when the process stops [13]. Time taken here gives us *Transfer Time*. **Restore**: Service is restored on destination using this com-mand [13]. Time taken in this phase is called *Restore Time*.

We performed the migration of different sizes of processes across two virtual machines (VM1 and VM2) on the same host. Virtual machines (Ubuntu18.04) run on Windows 10 operating system.When Live Migration triggers, VM1 starts saving process states and transfers them to VM2. It continues until two consecutive process states have the same hash. When the process has made no change, CRIU generates the same files. The size of such files is mostly in kBs. So, the process gets stopped on the source-side, once two consecutive process states obtain the same hash value. We transfer the last process state to VM2, and then the process is restored using this on VM2.

VII. RESULT ANALYSIS

Figure 3 shows Downtime that we get by Simple Migration (migration method used in the Existing approach), and Fig- ure 4 shows the Downtime of Live Migration. In comparison to Simple Migration, our approach of Live Migration gives lesser Downtime.

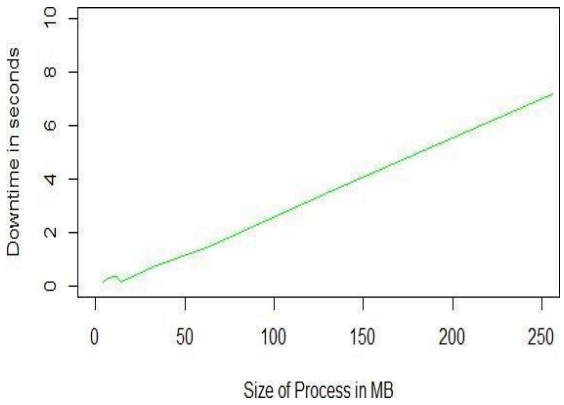

Fig. 4. Downtime caused by Live Migration

Figure 3 and Figure 5 are for the same events (Simple Migration), but Figure 5 shows the breakup of time taken by different parts of Simple Migration. We can see that as the size of the process increases, so does the dump time (because of the increase in dump size). It increases the transfer time of the file, which in turn increases the Downtime for the application. Similarly, Figure 6 shows the breakup of time taken by different parts of Live Migration. In this approach, the amount of data transferred during Downtime stays the same irrespective of program size, which results in Downtime also stays the same.Our results show that Downtime caused by Live Migration remains almost constant, at approximately 0.2 s. For small processes, Downtime caused by Simple Migration remains around 0.2 s. But for the process sizes more than 25 MB it increases as shown in Table II.

TABLE II
DOWNTIME

Process Size (in MB)	Time taken by Simple Migration (in seconds)	Time taken compared to Live Migration (x times)
32	0.7	3.5
64	1.5	7.5
128	3.4	17
256	7.1	35.5

VIII. CONCLUSION

In this paper, we concluded that breaking up the Stateful Process transfer into stages results in a more scalable ap- proach. After stopping the process,

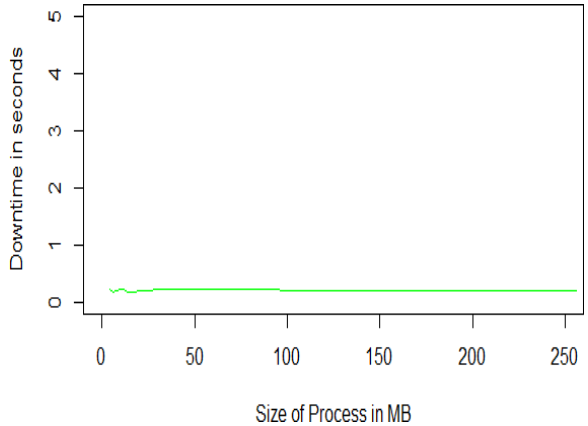

Fig. 4. Downtime caused by Live Migration

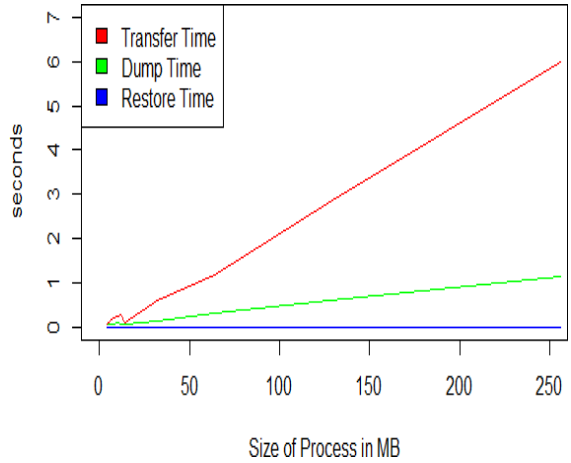

Fig. 5. Dump, Transfer and Restore Time caused by Simple Migration for processes with different sizes

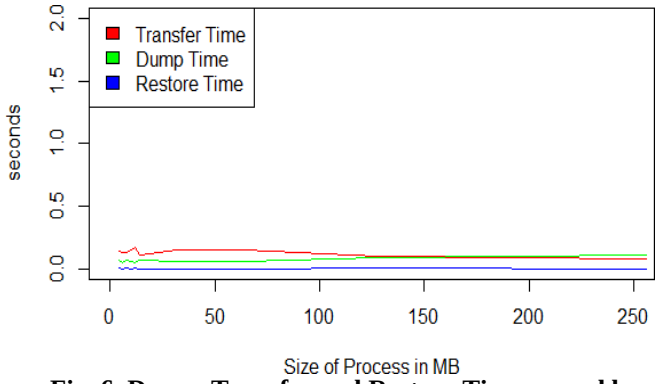

Fig. 6. Dump, Transfer and Restore Time caused by Live Migration for processes with different sizes

the size of the processstate, that needs to be transferred, stays constant with the increasing size of the process. This new approach results in the constant transfer time with respect to the size of the process. The increasing transfer time with the increase in the process size was the main limitation in the Stateful Process transfer approach that we improved upon. The Downtime by our approach is shorter than the existing technique by 6.9 s for 256MB process, and this difference will keep increasing for the large size of processes.The crucial point in our approach is the decision criteria of when to stop the process on the local server and get it ready for the migration. These criteria can be different for a different application. The criteria used in this paper works well for the applications which stay idle frequently. The criteria need to be changed for processes that see continuous activity, be it low or high, to keep the downtime to a minimum. The focus of future work can be the study of how different criteria affect different sets of applications.

REFERENCES

1. Hou X, Li Y, Chen M, Wu D, Chen JD, S . Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures. IEEE Transactions on Vehicular Technology. 2016;65(6):3860-3873.
2. Cuervo E, Balasubramanian A, Cho DK. MAUI: making smartphones last longer with code offload. Proceedings of the 8th international conference on Mobile systems, applications, and services pp. 2010:49-62.
3. Satyanarayanan M. The Emergence of Edge Computing.Computer. 2017;50(1):30-39.
4. Wang J, Pan J, Esposito F, Calyam P, Yang Z, Mohapatra P. Edge cloud offloading algorithms: Issues, methods, and perspectives. ACM Comput. Surv. 2019.
5. Ha K. Adaptive VM Handoff Across Cloudlets. 2015.
6. Machen A, Wang S, Leung KK, Ko BJ, Salonidis T. Live Service Migration in Mobile Edge Clouds. IEEE Wireless Communications. 2018;25(1):140-147.
7. Andersen T. Container Migration. 2014.
8. Randazzo A, Tinnirello I. Performance Analysis of Memory Cloning Solutions in Mobile Edge Computing. Fifth International Conference on Internet of Things: Systems, Management and Security. 2018:256-256.
9. Ma L, Yi S, Li Q. Efficient service handoff across edge servers via docker container migration. Proceedings of the Second ACM/IEEE Symposium on Edge Computing. 2017:11-11.
10. Horii M, Kojima Y, Fukuda K. Stateful process migration for edge computing applications. IEEE Wireless Communications and Networking Conference (WCNC). ;2018:1-6.
11. https://www.criu.org/Main_Page.
12. https://www.criu.org/Live_migration.
13. https://www.criu.org/Iterative_migration.

AUTHORS PROFILE

Harkiranpreet Kaur is currently pursuing M. Tech. degree at Guru Nanak Dev University, Amritsar and will be graduating in 2020. She is Bachelor of Technology degree holder from Computer Engineering and Technology department of Guru Nanak Dev University in 2018. She has a strong interest in the field of Computer Science.

Kiranbir Kaur obtained M. Tech. degree from Guru Nanak Dev University, Amritsar, India. Presently she is Assistant Professor at Computer Engineering and Technology department of Guru Nanak Dev University, Amritsar. She has a specialization in Cloud Computing. Her best publications are 'Kiranbir Kaur, Gurinder Kaur(2011)," Cloud Computing : Models" UGC sponsored National Seminar on Grid Computing', 'Kiranbir Kaur and Inderdeep Kaur(2010) , "Assessing Software Quality" in" Chal- lenges in Emerging Computer Technologies" at Rayat and Bahra Institute of Engineering and Bio-Technology, Sahauran'. She is a Gold Medalist in M.Tech. from Guru Nanak Dev University (2008).