

Container live migration in edge computing: a real-time performance amelioration

Dhuha Basheer Abdullah, Wael Hadeed*

Department of computer science, University of Mosul, 41002, Mosul, Iraq

ABSTRACT


A new trend of advanced applications with high demands has emerged in recent years. Though cloud computing provides ripe management services with ubiquitous abilities, new needs and workloads decreed by new services tend to unmask their deficiencies. Edge computing is a new type of computing that brings cloud services closer to customers. In addition to that, edge computing reduces client/server latency significantly. The services must work on edge nodes that are physical as near to their customers as reasonable to achieve slight latencies. As a result, when a client relocates, a service should migrate across edge nodes to preserve proximity. Besides, migration of containers between edge nodes allows for many emerging use cases, reduces back-to-the-cloud, and optimizes resource management (for example, e-learning systems). In this paper, an algorithm for managing container execution has been proposed. A set of constraints are considered when migrating containers between nodes, such as resource availability, deadline time, and nodes location. When an event occurs, the container must be migrated from one node to another closest/best possible node is searched. The container live migration is used to get the best possible response time, reduce server return, and better manage resources.

Keywords: Edge computing, Container, Docker, Optimal decision, Live migration.

OPEN ACCESS 

Received: May 10, 2021
Revised: February 6, 2022
Accepted: February 19, 2022

Corresponding Author:
Wael Hadeed
wael.hadeed@uomosul.edu.iq

 **Copyright:** The Author(s). This is an open access article distributed under the terms of the [Creative Commons Attribution License \(CC BY 4.0\)](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted distribution provided the original author and source are cited.

Publisher:
[Chaoyang University of Technology](https://www.cau.edu.iq/)
ISSN: 1727-2394 (Print)
ISSN: 1727-7841 (Online)

1. INTRODUCTION

In the last few years, a modern trend of advanced applications with a strong desire for Quality of Service (QoS) has evolved. Cloud computing is a technology that has been widely adopted over the last decade and depends primarily on the centralization of computing and data resources so that hand-out end users can use them on-demand. Services can be supplied by large data centralized centres located far away from users. Consequently, with the connection to remote services, a user may experience long latency. In recent years, significant progress has been made in bringing cloud services closer to customers, resulting in improved reliability and faster access (Salaht et al., 2020; Wang et al., 2018).

Modern information and communication technologies strategy aims to bring cloud processing as close to data sources as possible, leading to the emergence of novel computing paradigms such as edge, fog, and mist computing. Edge computing is a novel computing model that brings services cloud closer to end-users. Edge computing, among other benefits, exceptionally reduces client/server latencies. Every day, the number of real-time apps grows. These applications necessitate the availability of computational resources close to the equipment (Ketu and Mishra, 2021; Vasconcelos et al., 2019).

To achieve such minimum latencies, services must operate on edge nodes that are physically as close as feasible to their customers. As a result, when a client moves, a service must migrate across edge nodes to look after closeness. By managing resource allocation in the network edges and prioritizing time through load balancing,

mobile edge computing appears to be a viable way to reduce network operational expenses while also improving network node quality of service (QoS). Furthermore, service migration has a lot of potential for addressing the difficulties in determining when and where these services are migrated due to user mobility and demand changes (Wang et al., 2018; Benomar et al., 2020).

Edge computing is a divided, scaled-down version of cloud computing that can be used as a replacement for cloud computing while also providing extra benefits (Ngo et al., 2020). Virtualization allows cloud and edge computing to offer a single user an isolated computing/storage resource. As a result, a user can delegate computationally intensive activities to a dedicated virtual machine or container that runs on edge computing. Though cloud and edge computing can be used interchangeably in some applications, edge computing offers significant benefits due to its proximity to users. There are few security and privacy issues, reduced bandwidth, and fast response times because network transmission occurs right in front of customers (Puliafito et al., 2019).

The edge computing stand uses Docker containers (a type of software that can virtually package and isolate applications for deployment) to supply the required isolation of applications. Isolation is done by detaching their execution from external stimuli and obtaining multi-tenancy by reusing applications across different containers using virtualization (Kim et al., 2021). The service migration design can fully use the abilities of edge nodes abilities and decrease system downtimes by raising the fault tolerance. Docker container migration keeps the system running when a container stops responding due to overloading or failure. Docker container migration essentially pauses a live container in one location and then restores it to a new location with all of its data. During the live migration, users receive a connection interrupted notification, and their data is restored to its previous condition (Kaur and Kaur, 2020).

Containers can be moved in real-time from one physical hardware node to another within a data center, resulting in minimal downtime. In addition, the load can be rebalanced by migrating containers from one hardware node to another with live migration (Govindaraj and Artemenko, 2018; Ma et al., 2017).

In recent years, there has been a surge in interest in technologies that enable the integration of edge computing with containers, including several methodologies for dealing with remote infrastructure management and provisioning. Some concerns have been addressed in the literature in this context.

Maheshwari et al. (2018) proposed an approach to container migration by implementing a comprehensive system using a container hypervisor called Linux Container Hypervisor (LXD); then, he evaluated the container migration model based on real-time applications and took the example of license plate recognition based on

the mobile edge cloud. The evaluation measured quality-of-experience (QoE) and network efficiencies such as average system response time and relay cost for various loads, computing resources, bandwidth, and user latency. The author proposed a distributed resource migration algorithm and compared it with alternative techniques, and the results were satisfactory. Moreover, Linux containers are very popular because they have different advantages over virtual machines (VMs) and take advantage of the micro-service style software development mechanism. That is done by designing applications as independently deployable services. However, in the event of an attack or a resource issue, these applications do not enable container migration.

Dhumal and Janakiram (2020) suggested the C-Balancer system, which provided scheduling work for placing containers in the best possible way. C-Balancer works to improve containers' performance in terms of resource use and productivity; by experimenting with the proposed model, the maximum improvement in performance and variance in resource use has been achieved.

Another study by Benomar et al. (2020) proposed an intermediary cloud system using a set of possible tools to deploy and manage containers within the Fog and Mist cloud computing layer. The author considers the edge cloud computing layer to be peripheral; thus, it is possible to use other cloud layers for deployment and container management. The author used a set of tools, including the cloud industrial intermediary program OpenStack, in addition to Stack for Things (S4T). Furthermore, the migration of virtual machines and containers in dynamic resource management is a key aspect in lowering data center operating expenses by lowering energy consumption and, as a result, lowering the environmental effect. Smimite and Afdel (2020) proposed a dynamic strategy for how RAM is used for the host and virtual machines in the data center to prevent needless power usage. When the suggested model is compared to other techniques, the findings demonstrated that using containers instead of virtual machines saves energy usage and migration time, which affects the quality of service (QoS), and reduces service level agreement (SLA) violation. In the same context, the mobile node represents a significant burden for systems that rely on edge computing and container live migration. Addressing this type of problem is complex. Several authors have gone down their research path to address this condition. For instance, Ma et al. (2017) developed a framework that enhanced service handoff among edge offloading servers by exploiting Docker containers' multi-tiered storage architecture, improving migration performance, which was presented. This system enabled the edge computing platform to provide handoff services continuously in a short time by eliminating unnecessary transfers while supporting the mobility feature only when the migration process starts. After the experiment, the author showed that the handoff time decreased significantly.

According to the description above, several researches on container technology and edge computing were used in different scenarios. However, few studies evaluated the network's performance based on live container migration technology with edge computing. As a result, the current study helps assess how well the network performs when certain events, such as energy conservation, resource allocation, and fault tolerance, occur at specified periods. Furthermore, this work is also an extension of a work that was done in the previous studies by evaluating the performance of live migration and indicating the ability of the proposed algorithm that is based on real-time to choose the best/closest possible node and to exceed the failure case in the event of a disconnection.

2. THE PROPOSED METHOD

The work's main part is forming a group of nodes. The nodes are connected, specifying the IP and port of each node. Fig. 1 shows the main structure of the system. Also, the available resources (processor and memory) for all nodes are described.

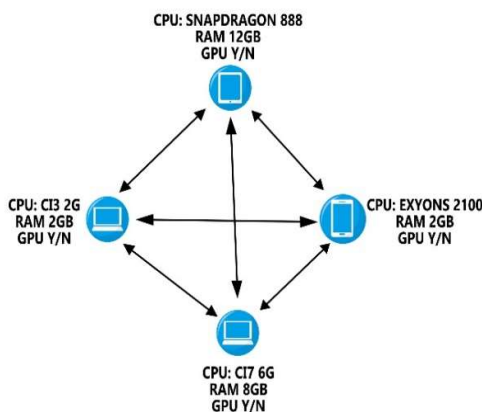


Fig. 1. Network nodes

An algorithm based on real-time technology and edge computing has been proposed to achieve the best possible way to communicate between nodes. Through cooperation between the nodes and the transfer of parts of the implementation between the nodes, depending on the selection of the best/nearest node. The main goal is to achieve a set of criteria that give strength to the system with perfect quality of service. The criteria that have been set are:

1. Container live migration: Use container live migration to reduce the time required for processing. Using CRIU Tool to continue the work of the container when it moves to another node from the point at which execution stopped. The ability to monitor and analyze physical resource usages, such as memory, processor, and networking, is enabled by migrating containers into a controlled environment.

2. Load Balancing: Supporting balance in the system even when there is a discrepancy in the physical parts of the nodes that make up the system by distributing containers on the nodes to ensure balance.
3. Maintenance: If there is maintenance at a particular node, the processing part of that node can be moved to another node.
4. Energy-saving: Energy consumption is a major challenge in edge computing systems that are resource-constrained. Saving resources often requires energy, especially for peripheral devices, so as much as possible to control energy consumption in this type of system.
5. Resources management allocation (dynamic run time): The use of the container and its transfer between nodes always needs to reserve resources. Resources must be available when the container moves to the leaf node. Controlling resources is one of the most important problems facing systems. Docker Swarm tool provides a convenient solution for controlling resource reservation; by allowing a node to manage containers and spread them on multiple host nodes.
6. Fault tolerance: For systems with multiple nodes, fault tolerance is a challenge. Although the behaviour of systems generally assumes the presence of faults during work, it must be taken into account that there is a tolerance for these errors. Improving the state of the terminal nodes is one of the solutions to take quick actions that reduce the impact of errors.

System Model

Initially, when the network is configured, each node will be permitted to access all network node resources. Access permission enables the network nodes to use the resources available on the node while adhering to what has been specified by Docker Swarm for all nodes. Docker Swarm uses to ensure that all nodes store the same consistent state. Thus, when a failure occurs, any node can restore balance and stability to the system. When an event (time priority, load balancing, node under maintenance, etc.) occurs at the current node and the node has a container in execution, the current state of the running container will be stored. Next, the search for the appropriate node to migrate the container begins. The update of nodes' information is done by Algorithm 1.

The proposed model targets edge computing. Until the computation is away from the server, part of the processing operations is transferred to the edge nodes by giving them some permissions. Events are periodically evaluated for all nodes to determine how active and properly the system behaves.

The system is divided into two parts: the first one is the Node Examination Unit (NEU). NEU is used to check the status of each active node, its resource utilisation, and the discretionary time of the node in an execution state. The process of node checking is done periodically by selecting

a release time and deadline for each execution cycle in a node in the event of no response (more than a period was taken). Second, Node Selector (NS). A container must be relocated to another node whenever a specified event happens. The list of nearest/best nodes is searched in the Nodes List (NL), as shown in Algorithm 2. A set of constraints are taken into account to select the appropriate node. The priority is the availability of the required resources, the state of the overload, and the proximity of the current node to the node that will be selected take precedence.

Algorithm 1: Node Examination Unit / Node List Checking

Input: Node List (NL)
While Node Status is Available **do**
 - **Determine** Deadline Time
 - **Check** Node Resources Available
 - **Check** Node Container Available
 - **Check** Node Location
 - **If** Container in Execution **do**
 Check Time Established
Output: New Node List (NL)

Algorithm 2: Node Selector (NS)

Input: Node List
For l : Node List (N) **do**
 - **Determine** Deadline Time
 - **Check** Node resources available
 - **Check** Node Location
 - **If** Node List (n) == Best **do**
 Candidate Node = Node List (n)
 Calculate Node Distance
Output: Candidate Node (NodeIP, NodePORT, NodeDIS)
 Start Container Live Migration

In Algorithm 2, the (Best) state means the following cases occur: First, the Deadline time of the Candidate node allows execution of the remainder of the container. Second, there are enough resources to complete the container's execution.

The list of nodes is sampled in Table 1, which illustrates the state of each node at the moment. When a specific event is received to evacuate node 1, the live container migration process begins to migrate the implementation of the Con1N1 container to another node. Using algorithm number 2 to find the best/nearest node. There are three main types of synchronization mechanisms (cold, pre-copy, and post-copy). The best one that applied in our work was the pre-copy migration. This mechanism states that the execution will not stop while searching for a candidate node for migration. Pre-copy migration takes shorter to migrate than post-copy and cold migration. That is because pre-copy migration does not convey the whole state of the source container to the destination host. Instead, when the migration process begins, it pre-dumps a portion of the container's state, often its memory pages, and transfers it to the destination host, leaving the source container alive and well. As a result, the state of the source container may yet be updated while the pre-dump state is being transmitted, and any state of the source container that is modified over the transfer is identified as modified to avoid any obsolete or conflicted states. Fig. 2 shows the Pre-copy migration.

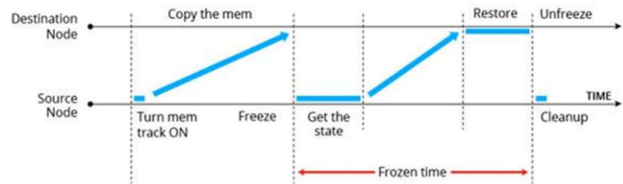


Fig. 2. Pre-copy migration

$$a = \sin^2\left(\frac{Lat1-Lat2}{2}\right) + \cos(Lat1) * \cos(Long2) * \sin^2\left(\frac{Long1-Long}{2}\right) \tag{1}$$

$$c = 2 * atan2(\sqrt{a} * \sqrt{(1-a)}) \tag{2}$$

$$d = 6371e3 * c \tag{3}$$

Table 1. System snapshot for four nodes

Node name	Information Description				
	NodeDL	ResAva (usage)	ConAvaQ	NodeLoc	EsConEx
Node 1	12:00:00PM	CPU: 33% RAM: 35%	Con1N1	Latitude: 36 22 44 Longitude: 43 08 37	120 sec
Node 2	1:00:00PM	CPU: 30% RAM: 40%	Available	Latitude: 36 23 10 Longitude: 43 08 51	-----
Node 3	12:00:00PM	CPU: 30% RAM: 40%	Available	Latitude: 36 21 56 Longitude: 43 05 12	-----
Node 4	2:00:00PM	CPU: 40% RAM: 50%	Con1N4 Con2N4 Con3N4	Latitude: 36 21 18 Longitude: 43 09 01	200 sec

Note: NodeDL: Node Deadline, ResAva: Resources Available, ConAvaQ: Container Available Queue, NodeLoc: Node Location, EsConEx: Estimated container execution time.

The distance is calculated by taking the nodes' geographical location (Latitude, Longitude), noting the equations 1, 2, and 3 (Gery, 1997) to calculate the displacement between the nodes.

After selecting the appropriate node through the search algorithm, the communication between the current and selected node via the IP and port is configured. Finally, the live migration is carried out with the help of CRIU's features. Fig. 3 illustrates the CRIU principle. The CRIU tool depends on several serial points starting from: first, sending an ACK packet from source to destination node; the goal is to give synchronization between them. Second, dump its current state as a set of files on disk.

Third, copy the dump file from the source to the destination node. Fourth, send an ACK confirmation packet from the source to the destination node. Fifth, freeze a working container. Sixth, resume the frozen container in the destination node. Seventh, stop and then destroy the container in the source node.

Initially, it is considered within the nodes list. Then, priority shall be given to effective contracts with no execution state. Another point is the inventory of nodes that contain the required resources. Then the displacement is calculated using equation (1 2 3).

Haversine's formula is used to find the distance between two points on a spherical surface by taking the latitude and longitude in degrees on the Earth. In equation 1, (Lat1, Long1) represent the coordinates of the first point, and (Lat2, Long2) represents the coordinates of the second point. Finally, (d) represents the distance in kilometers between two points in equation 3. Table 1 shows that node 2 and node 3 are in the "Available" state. Thus, the displacement is calculated and was (0.8753 km) between node 1 and node 2; and (5.276 km) between node 1 and node 3. So the migration process for the container will start at node 1 to node 2. CRIU technology was used to move the container. Fig. 3 shows the CRIU migration method for the container.

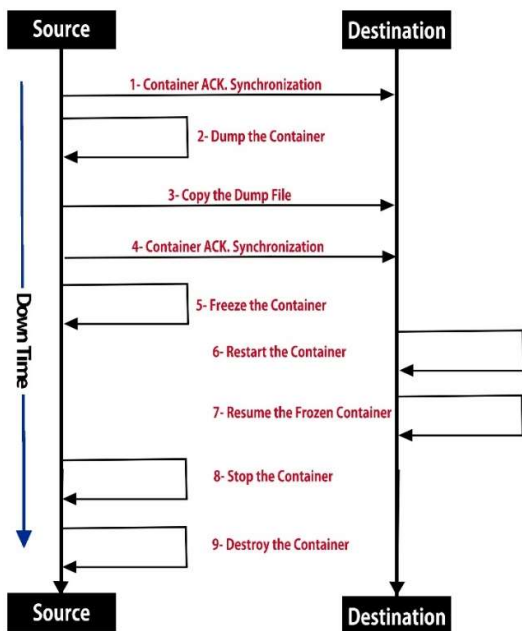


Fig. 3. CRIU technique

3. RESULTS AND DISCUSSION

The performance evaluation of the proposed algorithm was done based on the formation of four nodes within a virtual environment with the identification of resources and IP addresses, as shown in table 2 and Fig. 3. In table 1, the container migration process started from node 1 when an event occurred. Periodically, Algorithm 1 updates the list of nodes in the network. During implementation, several periods (1, 2, and 5 minutes) were tried to find the best possible period. Then, through Algorithm 2, the best/closest node is searched in the list of network nodes.

Table 2. Nodes description

VM name	Resources	IP Address
Vm1	-2 CPU core i5 -2GB RAM -12MB GPU	192.168.30.11
Vm2	-1 CPU core i5 -1GB RAM -12MB GPU	192.168.30.12
Vm3	-2 CPU core i5 -2GB RAM -12MB GPU	192.168.30.13
Vm4	-2 CPU core i5 -1GB RAM -12MB GPU	192.168.30.14

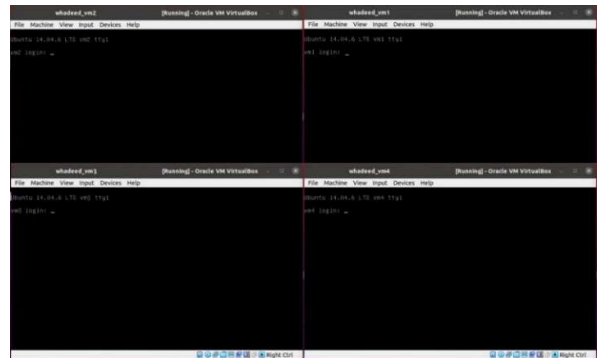


Fig. 4. Nodes virtualization

A container can have its whole state stored in a disk file because it is an isolated entity; this process is called checkpointing. Then, using that file, a container can be restarted. The solution takes advantage of Docker, which comes with the CRIU checkpoint tool, to checkpoint a task running in a container while performing a migration. Using CRIU technology, it is possible to move the container (Con1N1) currently executing on node 1 to node 2. CRIU is used to create a checkpoint and restore it. The container migration process starts after selecting the source and destination node. The migration process starts by opening

a communication channel between node 1 and node 2, depending on the IP address (192.168.30.12) and port number (8080) of node 2. Next, an ACK., the packet is sent to get synchronization between the two nodes. Then pause, dump is created to execute container (ConIN1) at node 1. Then the process of copying the dump file to the host node and starting to resume the work of the container (ConIN1) at the checkpoint at which the stop occurred in node 1. Fig. 5 shows the usage percentage per node.

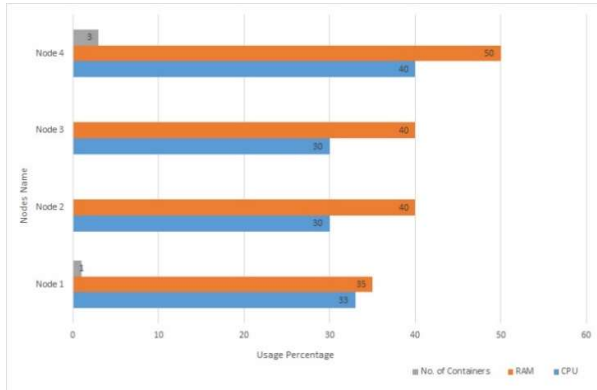


Fig. 5. Percentage usage per node

After container migration is complete, the destination node of the migrated process restores all container operations to their original state and sends resume messages. Resume messages are issued unconditionally if the partner node has not previously paused. Each pause and resume message contains information about the source and destination nodes. There will be no mistake regarding which containers must be paused or resumed if several containers migrate simultaneously. More precisely, each container in the system has its unique ID. Therefore, the system can differentiate the contents of each container. As a result, no issues in terms of containers' contents can happen when the migration is performed on many containers. If the migration process fails at any stage, the halted container will remain frozen and will not be able to communicate again. This scenario will not be frightening since the container will not be removed from the source node until the container's work has been resumed in the target node.

Calculating the response time using the proposed algorithm gave perfect results. Using the proposed algorithm to find the closest/best node for container migration, when experimenting, a good and optimal response time rate was obtained compared to the response time rate to find a suitable node regardless of the node's proximity to the parent node. One case that may give better results for response time is pre-scheduling or off-line scheduling, but its use is limited, does not support mobility features, and is not compatible with the principle of real-time. Fig. 6 shows the average response time of the proposed algorithm. By using the values in Table 3, each

case represents the response time using (proposed algorithm (Palg), directly algorithm (Dalg), and off-line scheduling (Offalg)).

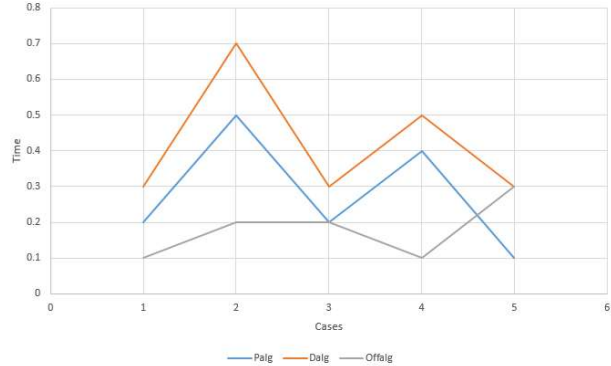


Fig. 6. Nodes response time

The proposed technique is compared to the no-migration situations in Fig. 7. Since container live migration is employed in load balancing, the average system response time is moderate compared to other alternatives. However, the average system reaction time is notable in the case of the nearest edge (nodes connect to the nearest available edge). The real-time approach utilized to track nodes is the cause behind this.

The above results are considered promising; however, it is needed to prove that these results are statistically significant. To this end, a regression model is built aiming to test the variations of these results using one-way Analysis of Variance (ANOVA) as follows: $lm(\text{performance} \sim \text{Palg} + \text{Dalg} + \text{Offalg})$, where performance is the dependent variable and Palg, Dalg, and Offalg are the independent variables. We also involved two hypotheses as follows:

Null Hypothesis: the means of all the algorithms are equal

$$H_0: \mu_{\text{Palg}} = \mu_{\text{Dalg}} = \mu_{\text{Offalg}}$$

Alternative Hypothesis: the means of the algorithms are not equal

$$H_1: \mu_{\text{Palg}} \neq \mu_{\text{Dalg}} \neq \mu_{\text{Offalg}}$$

The confidence level we choose is 95%, which means the value ($\alpha = 0.05$). However, after implementing the model, we found a p-value of (0.001) lower than the significance level; This means we cannot accept the Null Hypothesis of equal means. Therefore, the means are different and the results obtained are statistically significant.

The topic of container live migration in edge nodes have been considered before. For example, Maheshwari et al. (2018) proposed an approach to container migration, a set of techniques for efficient VM live migration on edge, while Dhumal and Janakiram (2020) suggested the C-Balancer system, which provided scheduling work for placing containers in the best possible way. Finally,

Table 3. Average nodes response time

	Case 1	Case 2	Case 3	Case 4	Case 5	AvRsT	Variance
Palg	0.2	0.5	0.2	0.4	0.1	0.28	0.027
Dalg	0.3	0.7	0.3	0.5	0.3	0.42	0.032
Offalg	0.1	0.2	0.2	0.1	0.3	0.18	0.007

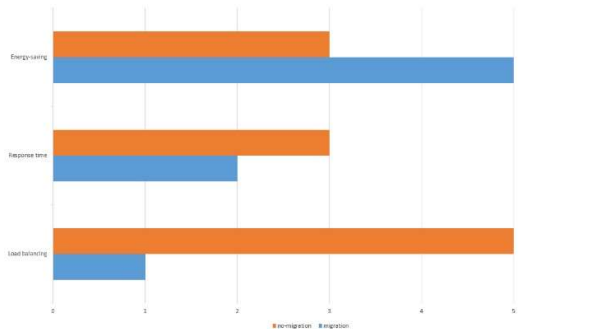


Fig. 7. Migration and no-migration comparison

Benomar et al. (2020) proposed an intermediary cloud system using possible tools to deploy and manage containers within the Fog and Mist cloud computing layer.

The difference between this work and the other works in the literature is that none of the previous works used metrics such as deadline time, resources available, overload, and node's place to choose the best node. Moreover, they did not take these metrics together; choosing the appropriate node is more accurate using the proposed algorithm, which makes it the first to address the problem.

Finally, some additional papers, such as those (Smimite and Afdel, 2020; Kotikalapudi, 2017), looked at migration as scheduling, mapping, and orchestration problem. Although the suggested approach does not address these issues, it can be used to exploit or leverage such works.

4. CONCLUSION

Although there is a waste of time during the live container migration process, especially for systems that depend on the principle of real-time, the use of direct migration gives greater flexibility with the stability of the system in general and the presence of good quality of services in addition to ensuring that work does not stop when an emergency occurs in one of the network nodes. Although dealing with moving nodes complicates the calculations to find the nearest suitable node. Nevertheless, the algorithm to find the best/nearest path was very effective. The proposed algorithm gives a set of initial solutions before the containers are migrated. The nodes list provides a complete knowledge of the network infrastructure, the most important of which are the available resources and the location of the nodes. Thus periodically provides information on all network nodes, which in turn gives planning for dynamic resource management. Getting optimal response time by sticking to

time slots and real-time constraints. There are challenges in protecting mobile containers between nodes; through unauthorized access, if it is to the network or nodes, protection must be provided to the data of all customers. We recommend this in future work.

ACKNOWLEDGMENT

The author is grateful to the Department of Computer Science / College of computer science and mathematics at the University of Mosul / Iraq for supporting me in completing this work.

REFERENCES

- Benomar, Z., Longo, F., Merlino, G., Puliafito, A. 2020. Cloud-based enabling mechanisms for container deployment and migration at the network edge. *ACM Transactions on Internet Technology*, 20. <https://doi.org/10.1145/3380955>
- Dhumal, A., Janakiram, D. 2020. C-balancer: a system for container profiling and scheduling. 1–10. <http://arxiv.org/abs/2009.08912>
- Gery, S.W. 1997. Direct fix of latitude and longitude from two observed altitudes. *Navigation, Journal of the Institute of Navigation*, 44, 15–24. <https://doi.org/10.1100/j.2161-4296.1997.tb01935.x>
- Govindaraj, K., Artemenko, A. 2018. Container live migration for latency critical industrial applications on edge computing. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2018-Sept(iii)*, 83–90. <https://doi.org/10.1109/ETFA.2018.8502659>
- H., N.S., Kumar, K.R.A., Shenoy, S.N., Rao, A.S., 2020. Data security in cloud environment based on comparative performance evaluation of cryptographic algorithms. *International Journal of Advanced Trends in Computer Science and Engineering*, 9, 4989–4997.
- Hadeed, W.W., Abdullah, D.B. 2021. Real-time based big data and e-learning: a survey and open research issues. *AL-Rafidain Journal of Computer Sciences and Mathematics*, 15, 225–243.
- Havanje, N.S., Kumar, K.R.A., Shenoy, S.N., Rao, A.S., Thimmappayya, R.K. 2022. Secure and reliable data access control mechanism in multi-cloud environment with inter-server communication security. *Suranaree Journal of Science & Technology*, 29.
- Kaur, H., Kaur, K. 2020. Live Migration of stateful processes across edge servers. *International Journal of*

- Recent Technology and Engineering, 8, 5207–5211. <https://doi.org/10.35940/ijrte.e9850.038620>
- Ketu, S., Mishra, P.K. 2021. Cloud, fog and mist computing in iot: an indication of emerging opportunities. IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India), 0, 1–12. <https://doi.org/10.1080/02564602.2021.1898482>
- Kim, T., Al-Tarazi, M., Lin, J.W., Choi, W. 2021. Optimal container migration for mobile edge computing: algorithm, system design and implementation. IEEE Access, 9, 158074–158090. <https://doi.org/10.1109/ACCESS.2021.3131643>
- Kotikalapudi, S.V.N. 2017. Comparing live migration between linux containers and kernel virtual machine: investigation study in terms of parameters. February, 42. www.bth.se
- Ma, L., Yi, S., Li, Q. 2017. Efficient service handoff across edge servers via docker container migration. 2017 2nd ACM/IEEE Symposium on Edge Computing, SEC 2017. <https://doi.org/10.1145/3132211.3134460>
- Maheshwari, S., Choudhury, S., Seskar, I., Raychaudhuri, D. 2018. Traffic-Aware Dynamic Container Migration for Real-Time Support in Mobile Edge Clouds. International Symposium on Advanced Networks and Telecommunication Systems, ANTS, 2018-(December). <https://doi.org/10.1109/ANTS.2018.8710163>
- Nagesh Shenoy H, K.R. Anil Kumar, Rajgopal K.T., Abhishek S. Rao. 2020. An Audit on cloud architectures addressing data privacy and security concerns. International Journal of Advanced Science and Technology, 29, 6373–6382. Retrieved from <http://sersc.org/journals/index.php/IJAST/article/view/20081>
- Ngo, M.V., Luo, T., Hoang, H.T., Tony Quek, Q.S. 2020. Coordinated container migration and base station handover in mobile edge computing. 2020 IEEE Global Communications Conference, GLOBECOM 2020 - Proceedings. <https://doi.org/10.1109/GLOBECOM42002.2020.9322368>
- Puliafito, C., Vallati, C., Mingozi, E., Merlino, G., Longo, F., Puliafito, A. 2019. Container migration in the fog: A performance evaluation. Sensors (Switzerland), 19, 1–22. <https://doi.org/10.3390/s19071488>
- Salaht, F.A., Desprez, F., Lebre, A. 2020. An overview of service placement problem in fog and edge computing. ACM Computing Surveys, 53. <https://doi.org/10.1145/3391196>
- Smimite, O., Afdel, K. 2020. Containers placement and migration on cloud system. International Journal of Computer Applications, 176, 9–18. <https://doi.org/10.5120/ijca2020920493>
- Vasconcelos, D.R., Andrade, R.M.C., Severino, V., De Souza, J.N. 2019. Cloud, Fog, or Mist in IoT? That is the question. ACM Transactions on Internet Technology, 19. <https://doi.org/10.1145/3309709>
- Wang, S., Xu, J., Zhang, N., Liu, Y. 2018. A survey on service migration in mobile edge computing. IEEE Access, 6, 23511–23528. <https://doi.org/10.1109/ACCESS.2018.2828102>