Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

Optimal Container Delivery for Mobile Edge Computing: Algorithm, System Design and Implementation

Dr. Amaresh Sahu[1], Rashmita Sahoo[2]*

[1] Associate Professor Dept. Of Computer Science and Engineering, NIT , BBSR*
[2] Assistant Professor,Dept. Of Computer Science and Engineering, NIT , BBSR
*amreshsahu@thenalanda.com*rashmitasahoo@thenalanda.com*

ABSTRACT Edge computing is a promising alternative to cloud computing for offloading computationally heavy tasks from resource-constrained mobile user devices. Placed at the edge of the network, edge computing is particularly advantageous to delay-limited applications for having a short distance to end- users. However, when a mobile user moves away from the service coverage of the associated edge server, the advantage gradually vanishes, increasing response time. Although service migration has been studied toaddress this problem focusing on minimizing the service downtime, both zero-downtime and the amount of traffic generated as a result of migration need further study. In this paper, an optimal live migrationfor containerized edge computing service is studied. This paper presents three zero-downtime migration techniques based on state duplication and state reproduction techniques, and then, proposes an optimal migration technique selection algorithm that jointly minimizes the response time and network traffic duringmigration. For validation and performance comparison, the proposed migration techniques are implementedon off-the-shelf hardware with Linux operating system. The evaluation results showed that compared witha naive migration, the optimal approach reduced the response time and network load by at least 74.75% and94.79%, respectively, under considered scenarios.

INDEX TERMS Container, docker, edge computing, implementation, live migration, optimal decision, service migration.

I. INTRODUCTION

We are living in an era where Internet of Things (IoT) devices in the vicinity of users continuously produce massive data [1]and process them [2] to enable user-centric applications to berunning on smart handheld devices. In particular, computa- tionally heavy tasks, such as big data analytics, virtual reality (VR)/augmented reality (AR), image processing, etc., are the key ingredients for the further success of such applications. However, neither IoT devices nor handheld devices suffice toprovide real-time interactions while running such computa- tionally heavy applications due to their limited resources [3].The need for enriching user experience has popularized cloudcomputing which allows low-power devices to offload com-putationally heavy tasks [4]. Centralizing a massive amountof IT resources, cloud computing can provision computing resources on demand, and adapt to varying service demand (e.g., auto scaling [5]). In addition, virtualization plays a keyrole in cloud computing to achieve high resource utilization,resource isolation and multi-tenancy [6].

To the resource-limited devices, the task offloading to cloud computing significantly helps to reduce the job processing time and the battery uses [1]. However, the large distance to end devices may incur high delay, making it unsuitable to delay-sensitive applications suchas healthcare, connected vehicles, AR, and surveillance/ monitoring [7]–[9]. To provide additional computing resources to resource-limited end devices and minimize the

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

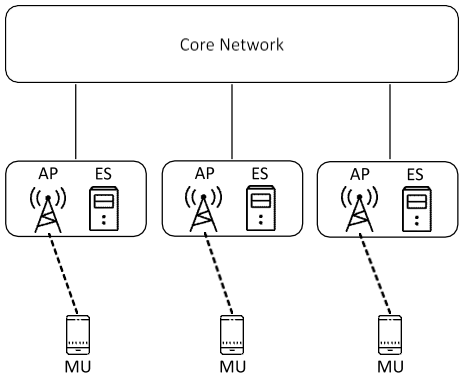


FIGURE 1. Mobile edge computing system where a mobile user communicates with its associated access point, and the offloading servicerunning on the co-located edge server allows a mobile user to offload its tasks to edge server.

latency at the same time, edge computing [1], [7], [9]–[14] has emerged as a promising alternative [15], [16]. In partic- ular, edge computing places computing/storage resources at the edge of the network. Thus, the reduced distance to end devices effectively decreases the network delay.

Edge computing is a distributed and downsized form of cloud computing, and thus, it can be an alternative to cloud computing [15], [16] with additional benefits. Both cloud computing and edge computing can provide an isolated com- puting/storage resource to an individual user with virtualiza-tion. Thus, a user can offload its computationally-heavy tasksto its dedicated virtual machine or container running on either cloud or edge computing. Although both cloud and edge computing can be used interchangeably in some applications,the short distance to a user in edge computing can bring manyadvantages. Reduced security/privacy threat, bandwidth cost and response time are some of such advantages due the fact that network transmission occurs only in the vicinity of users [10], [11].

In some literature, the definition of edge includes the end devices such as smart phones. However, in this paper, edge is assumed to be the end of the radio access network operated and managed by the network operator. Thus, the scenarios where the end devices participate in task offloading are not considered in this work. Also, the partial offloading case where a fraction of a task is offloaded to an edge server whereas the remainder is processed locally at the user device is beyond the scope of this paper.

The Fig.1 shows an example network diagram with edge computing. An edge server (ES) is attached to an access point (AP), and the users associated with an AP can offload their tasks to the co-located ES. Each user has its own service running on an ES, and due to virtualization and multi-tenancy, it does not violate the operation of other active serviceson the same ES. When a user moves out of the wireless coverage of the associated AP, the handoff procedure is trig- gered to transfer the connection of the user to another AP. To keep the service delay minimized while users movearound, transferring a service running on an ES to another has been proposed, called service migration. In particular, stateful and live service migration is challenging [11], [14] since it migrates a running service as it is without interrupting the ongoing service. In order to expedite the service migration process, the use of light-weight containerized virtualization has received much attention, such as Docker [17]. As reported in [18] and [19], Docker can achieve close-to native perfor- mance.

There have been a few studies proposing service migra- tion techniques, but the major limitation is that they didnot jointly consider the characteristics of the to-be-migrated service and migration technique to use. Since each service has different properties, to efficiently migrate containerized services, different migration technique should be chosen in an autonomous manner. Also, as it can be seen in Fig.1,the network traffic generated as a result of migration is injected to the core network, and it may cause network congestion if the traffic is voluminous. Thus, one should carefully design a migration method in order not to generatetoo much network traffic while keeping the migration time minimized.

In this paper, three live, stateful container migrations and an autonomous system that selects the optimal migrationare proposed. To the best of our knowledge, this is the first comprehensive study of the containerized service migration in the sense that 1) different migration techniques are consid-ered together to choose the optimal migration technique, and
2) the migration techniques as well as the optimal selection system has been validated by implementation. The major contribution of this study is summarized below.

- This paper proposes enhanced container migration tech- niques that migrate both the persistent files and volatile states (e.g., CPU context and memory state). In par- ticular, by introducing packet relay and replay buffer during migration,

the proposed techniques achieve zero-downtime during migration.

- This paper identifies the key characteristics of both three migration techniques and containerized services to be migrated. Then, this paper analyzes which technique outperforms the rest in which services in terms of migra- tion time and network load.
- This paper proposes an optimal migration selection method that minimizes both the expected migration time and the expected amount of traffic to be generated.
- This paper also proposes a low-cost optimal migration selection algorithm for for it to run in real-time.
- This paper proposes a system design to carry out the optimal migration selection and the chosen migration procedure in an autonomous manner.
- This paper introduces the practical and technical details on how to implement the migration techniques on the widely-used Docker platform.
- The proposed migration techniques and the automated optimal migration selection system are implemented on off-the-shelf computing devices.

- This paper presents an empirical evaluation results, and shows that the proposed algorithm can effectively and efficiently migrate containerizes services.

The rest of the paper is organized as follows. In Section II a summary of previous studies on containerized service migra-tion is presented, followed by a brief introduction to the backgrounds on the container migration on Docker. The pro- posed three different live container migration techniques are introduced in Section III, and then, the proposed optimal migration selection algorithm along with the overall system design is presented in Section IV. The following Section V explains the overall methodology used in this work. The empirical evaluation results are presented and discussed in Section VI, and finally, the paper is concluded by Section VII. The frequently used abbreviations and acronyms throughout this paper are summarized in Table 1.

TABLE 1. Frequently used abbreviations and acronyms.

AP	Access Point
ES	Edge Server
MU	Mobile User
QoS	Quality of Service
RTT	Round Trip Time
ES	Edge Server (or Edge computing Server)
ES(src)	ES from which the running container is migrated
ES(tgt)	ES to which the running container on ES(src) is migrated
DC	Proposed Differential-Copy migration technique
FC	Proposed Full-Copy migration technique
LR	Proposed Log-Replay migration technique

II. LITERATURE REVIEW AND BACKGROUNDS

In this section, previous studies on the containerized service migration are discussed, followed by a brief introduction to the backgrounds on the container migration on Docker.

A. RELATED WORK

Puliafito *et al.* [6] compares pre-copy, post-copy and hybrid migration performance in terms of migration time, service down time and the amount of transferred data. In short, pre-copy iteratively transfers most of the states (i.e., dirty pages) before stopping ES(src), while post-copy minimizes the amount of pre-copy and transmits dirty pages when it is requested at ES(tgt). The hybrid is the combination of both. The tools used in their paper are CRIU [20] for checkpointing, rsync for file transfer, and runC for container runtime. How-ever, the authors assumed that writing to disk is not allowed for some applications, and did not transfer the persistent files. Karhula *et al.* [21] demonstrated the migration of IoT edge functions using Docker and CRIU for checkpointing. How- ever, their proposed method is limited in that persistent files are not synchronized between ES(src) and ES(tgt), and there is no consideration on the changed states after the checkpoint has been made.

Nadgowda *et al.* [22] proposed a stateful migration on Docker. The proposed Voyager transfers the memory state to the ES(tgt). In addition, to minimize the service down- time during the local file system migration, a dual-band data transfer is proposed by using a network file system. However, if the network-attached storage is not available, the migrated service may

suffer from frequent faults for the files that are being copied in background, which may significantly degrade the quality of service (QoS). Dupont *et al.* [23] proposed a migration orchestration system, called Cloud4IoT. By using Docker and Kubernetes [24], the proposed system can per- form horizontal and vertical migration of IoT functions. Thelimitation in their work is the underlying assumption that ser-vices are stateless. Thus, Cloud4IoT does not transfer states,and it cannot be used when stateful migration is necessary.

Al-Dhuraibi *et al.* [25] proposed an automatic vertical scaling system for Docker containers, called ElasticDocker. Although it is different from containerized service migration, ElasticDocker does perform live migration when there is no resource left on the host for scale-up. The live migration in ElasticDocker first transfers the file system differences, andthen, transmits memory states. Their proposed live migrationis similar to `FC` that is proposed in this paper (see Section III for detail). The limitation of ElasticDocker is that it freezes the container at the final memory dump stage. If a user sendsrequests before the container on ES(tgt) starts, they may getlost, resulting in QoS decrease or state inconsistency.

Ma *et al.* [26], [27] proposed a container live migra-tion leveraging the layered storage of Docker. Their pro- posed method can be summarized in three steps. 1) Image layer synchronization: the different image layers are trans- mitted to ES(tgt). 2) Memory difference transmission: this stage transmits the checkpoint for consistent memory state.
3) Container stop and container layer synchronization: this step synchronizes the writable container layer, i.e., the user's modifications to files/directories are transmitted to ES(tgt). The limitation of this work is twofold. In the pre-dump con-tainer stage, memory snapshots are transmitted to all potential target servers, and it may incur unnecessary network load. Also, the later stage of migration terminates the ES(src) before ES(tgt) becomes ready to provide service. This may cause a short period of service interruption (i.e., downtime), or in the worst case, permanent service down when the ES(tgt) falls into faulty state.

Yu *et al.* [28] proposed a container migration with log- ging and replay. The proposed 3-stage migration runs as follows: 1) exporting and transferring the entire image includ- ing the container layer, 2) logging the changes while the image transmission is in progress and replaying the changes on ES(tgt), 3) stopping ES(src) and resuming ES(tgt). Although the migration scheme can migrate the persistent files, some memory pages are not to be migrated. This can be a limitation to stateful applications. A similar approach has been proposed for virtual machine (VM) environment. Liu *et al.* [29] proposed a VM migration scheme that first transmits the checkpoint of a running VM. Until the VMon the target host is in the consistent state with the source host, the system events are recorded and replayed at the target host. However, this approach does not consider trans-ferring the persistent files on the local file system of the

source host. In addition, in some applications logging- and-replay by itself can be a better solution than the com- bination of checkpointing and logging-and-replay.

As it can be seen in the previous literature, there are differ-ent ways of stateful migration: state-transfer or state-rebuildby logging and replay. Also, different services have different properties which will be discussed in the following sections. However, none of the aforementioned studies jointly consid-ers both to choose the optimal migration technique in terms of migration time and network load for minimizing service disruption and network congestion, respectively. This paperproposes three effective migration methods, and then, pro- poses to jointly consider the properties of different migrationtechniques and the service applications to be migrated in order to migrate services in an time and network-efficient manner. Furthermore, this paper presents an autonomous migration selection system that selects the optimal migrationgiven the extracted properties of the container to be migrated.

B. BACKGROUNDS: STATEFUL CONTAINER MIGRATIONWITH DOCKER

Virtualization is a key component to enable cloud/edge com-puting for high resource utilization, resource isolation and multi-tenancy. Among different virtualization technologies, containerization has become more popular in the edge com- puting domain for their high performance and light-weight nature [6], [30], [31]. Although Docker is not the only containerized virtualization solution (e.g., LXC [32] andOpenVZ [33]), it is studied in this paper for its widespread use and large market share, i.e., approximately 25% [34]. Also, it is shown that Docker containers operate at the close-to native performance even on single-board computers [35]. This section introduces some Docker features that play a key role in the proposed live container migration and its implementation.

A container is a complete runtime environment includingan application and its dependencies. In particular, Docker iso- lates resources between different containers by using *names- pace*, and *cgroups* to control/monitor resources such as CPU and memory [37]. By sharing the kernel, containers can be lightweight [25], which is a distinguishable feature from virtual machine software such as VirtualBox and VMWare (see [13], [37] for an in-depth comparison between containers and virtual machines).

A Docker container is created from a downloaded or cus- tom image which is a set of read-only layers constituting an application or service. When a container is started from an image, a writable layer, called container layer, is added as a top layer. Docker supports different storage drivers to store image layers (e.g., overlay2, btrfs and aufs), and overlay2is used in this paper as officially recommended [38]. The writable container layer is denoted by *upperlayer* in the over-lay2 driver, and it contains the differences or changes in thefile system (i.e., modified, added or deleted files/directories).In other words, the list or files and directories that are different from those in the image can be found in the upperlayerdirectory, which is also available by using the Docker `diff`
command.

Docker also records the container's `STDOUT` and `STDERR`as logs in a JSON format by default. The log file can be found in the *LogPath* directory of which path is located in the `docker inspect` result. Also, the log can be retrieved by using the `docker logs` command. This is particularly useful to trace the history of actions executed mostly by userand applied to the container.

CRIU is a utility to checkpoint and restore the state of a Linux process. CRIU captures memory state, process states,open files, network sockets and so forth, and dumps as a collection of files. The size of the checkpoint depends on thepagemap size of the process [22]. The Dockers experimentalcheckpoint function [39] relies on CRIU, and this play a keyrole in migration to let the volatile state of ES(tgt) be the same to that of ES(src).

III. PROPOSED STATEFUL LIVE MIGRATION

To achieve seamless, live migration that is transparent [8] to users, this section introduces the three enhanced migration techniques that are proposed in this paper.

A. ASSUMPTIONS

In the assumed system in this paper, an ES is co-located withan AP since deploying an ES at the AP reduces the service delay and avoids network congestion [7]. In this study, mobile users access the network via an IEEE 802.11 AP, whichis wire-connected to the core network. A user entering the coverage of an AP associates with the AP for wireless connec- tion, and it triggers a handoff procedure [40] if the user is in association with another AP at the moment. A user associated with AP-*n* receives computation offloading service from its dedicated container running on the ES-*n* which is co-located with AP-*n*. When a user handoffs to another AP, container migration is triggered and the user's container is moved to the new AP so as

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

to keep the network delay minimized.

When migration is triggered from ES(src) to ES(tgt),widely-available docker images, for example, those that are accessible in either public/private repositories, are assumed to be locally available (or pre-cached) on the ES(tgt) [36]. How- ever, in the case of containers running customizes images, this assumption does not hold, and it can be handled by using the proposed FC to be explained shortly. ES hosts have similar computing power, and the default resource-limitconfiguration [41] is applied to all Docker containers.

B. PROPOSED LIVE CONTAINER MIGRATION

The proposed live migration techniques synchronizes ES(tgt) with ES(src), and ES(tgt) will eventually have the same persistent state (i.e., files and directories in the file system) and volatile state (i.e., memory layout, network sockets, open files, etc.) as ES(src) has. The proposed three migration tech-niques can be roughly classified into two methods, namely *state duplication* and *state reproduction*. State duplication copies the state from ES(src) and then transfers to ES(tgt) so

that ES(tgt) can start from the most recent state of ES(src). On the other hand, state reproduction transfers an instruction execution log/trace to ES(tgt) so that ES(tgt) can reproduce a container with the consistent state to ES(src) by executing the instructions in the received log. Throughout this paper, the scenario in Fig. 2 is assumed as an illustrative example. A user associated with AP-1 is receiving offloading service from a container on ES-1. As a user moves away from AP-1, it re-associates with AP-2, which initiates the containerized service migration. To be specific, the container migration is triggered by receiving the corresponding message from the controller, which is explained in Section IV. The controller sends different messages to ES(src) and ES(tgt), namely, theidentification (i.e., IP address) of the ES(tgt) to ES(src) and the name of the base image to ES(tgt). The base image nameis not used in FC since it transfers the complete image, but in both DC and LR it can be used to check if the image is locallycached on ES(tgt).

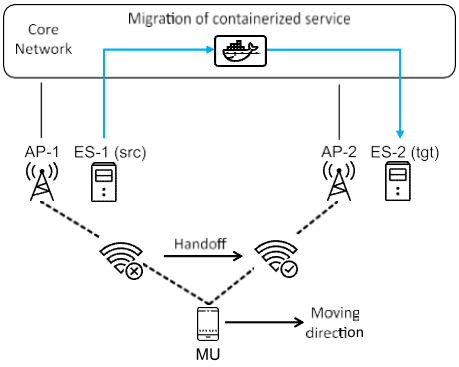


FIGURE 2. An example scenario that a user handoffs to AP-2, whichtriggers migration of the containerized service from ES-1 to ES-2.

The distinct features of the proposed migration techniquescompared to the previous works are as follows.

1) The proposed techniques migrate and synchronize both persistent files and volatile states.
2) By using the packet relay during migration, the pro- posed migrations can effectively prevent service outageand achieve zero-downtime.
3) By using the replay buffer during migration, the migrated container can synchronize its state with the migrating container without requiring any additional state transfer.

1) FULL-COPY MIGRATION, FC

FC is a naive state duplication method to be used mainly asa performance baseline in this study. When a container on ES(src) at AP-1 is migrated to ES(tgt) at AP-2 all container layers (i.e., writable layers and read-only layers) as well as execution state are transferred to ES(tgt) so that a container with the consistent state can be started on ES(tgt) only by using what has been transmitted.

The overall procedure of FC migration is illustrated in Fig. 3. When a migration is triggered, the to-be-migrated

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

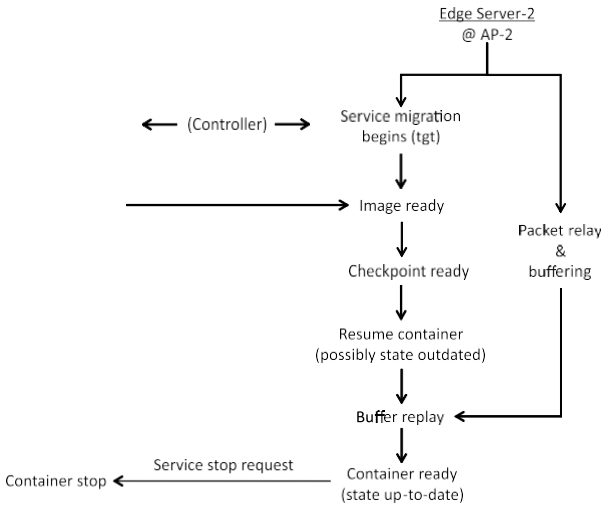


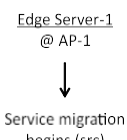
FIGURE 3. Flowchart of the proposed Full-Copy migration: FC migrates the entire container image as well as the checkpoint so that ES(tgt) can resume the service only with what has been transmitted. The buffering and packet replay is used to trace and replay the state changes occurringduring migration.

container on ES(src) is exported and saved as a new image including both writable and read-only layers. Docker `commit` and `save` commands are used for this task. Then, the new image including the persistent state of the containeris transferred to ES(tgt). The following step is to transferthe execution state, and it can be done by checkpointing by using `checkpoint create` command and sending it to ES(tgt). All file transmissions are carried out by a secure copy protocol tool, `scp`.

In this work, when exporting and checkpointing the container, the container is configured to keep running by using `--pause=false` and `--leave-running=true`option, respectively, so that it can continuously provide ser-vice to the user until its replica becomes ready on ES(tgt). From the moment MU handoffs to AP-2 to when the con- tainer becomes ready on ES(tgt), a non-negligible amount oftime passes, during which the container's state might change.To trace such changes, the proposed migration employspacket relay and buffering technique.

Right after a user handoffs to AP-2 its serving containeris still operating on ES(src), while the user accesses AP-2 for communication. Until the container on ES(tgt) becomes ready for service, the proposed migration allows AP-2 to relay the user requests to AP-1 so that ES(src) can pro-vide service. In the meantime, AP-2 buffers the same user requests locally. When the container on ES(tgt) starts, all buffered requests are replayed on the container. As soon as thebuffer becomes empty, ES(tgt) notifies AP-2 of its readiness. AP-2, then, stops packet relay and buffering so that its local ES, i.e., ES(tgt), can provide service to the user. Also, AP-2 sends service stop request to ES(src) so that ES(src) can stop and release the container resource. ES(src) may delay the container stop for a certain amount of time, called grace period, if there are waiting jobs in its queue.DIFFERENTIAL-COPY MIGRATION, DC

`DC` does not transfer the read-only layers, and thus, it can achieve an efficient state duplication compared to `FC`. The reduced amount of data transmission not only shortens the migration time, but also alleviates network congestion com-pared to `FC`. As aforementioned in Section II-B, a containerconsists of a writable layer at the top and read-only layers below. If the base image from which the container has startedis locally cached on ES(tgt), the consistent state in termsof the persistent state can be achieved only by migratingthe writable layer. The checkpointing is also used in `DC` to transfer the volatile state to ES(tgt). The overall procedure of `DC` is illustrated in Fig. 4 assuming that when migrating a containerized service, the base image is pre-cached on ES(tgt) or it can be pulled from nearby repository instantly.



Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

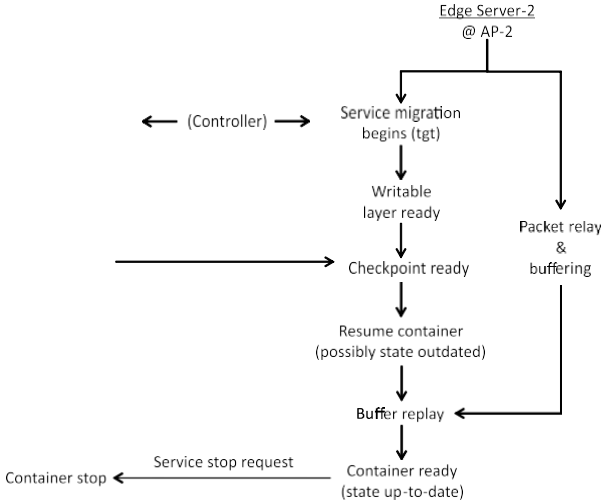


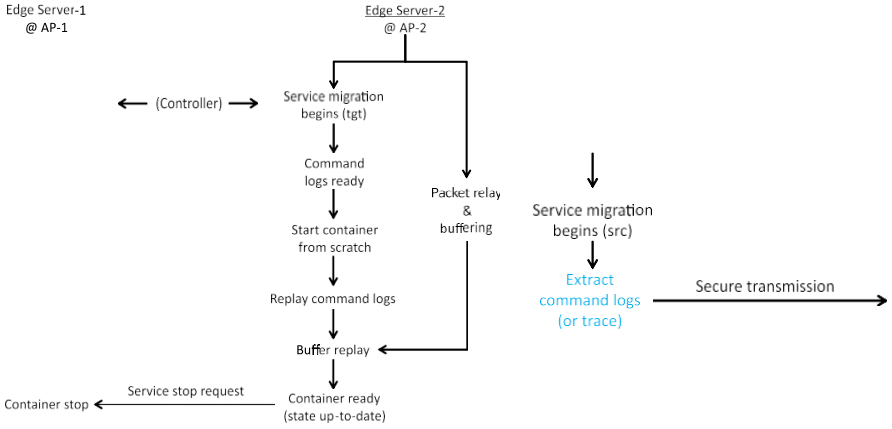
FIGURE 4. Flowchart of the proposed Differential-Copy migration:
DC migrates the writable layer as well as the checkpoint so that ES(tgt)can resume the service in a consistent state with reduced data transmission. The buffering and packet replay is used to trace
and replay the state changes occurring during migration.

When the migration is triggered by receiving the corre- sponding message from the controller, ES(src) extracts the file system changes (i.e., writable-layer contents), and ES(tgt) checks if the base image is locally available. The nameof the base image is known to ES(tgt) by Controller. The writable-layer contents can be extracted in two ways. One isto list the changes by using `docker diff` command and extract them with `docker cp`. The other is to locate the path in which the changes are saved on the ES host's file system. The path is identifiable from the `docker inspect` result by using the value of the `UpperDir` label. The former is *safe*but it requires parsing the `docker diff` result, and thus,the proposed `DC` uses the later for efficiency.

After transmitting the writable-layer contents, ES(src) makes checkpoint and then transmits it to ES(tgt). Upon receiving both, ES(tgt) resumes the container from the up- to-date state. Any changes that are not included in either thewritable layer or the checkpoint are handled by the packet replay and buffering method.`DC` is similar to `FC` in the sense that both transfer stateto ES(tgt). However, `FC` transfers much larger data, and asa result, it longer and incurs larger network load than `DC`.

2) LOG-REPLAY MIGRATION, LR

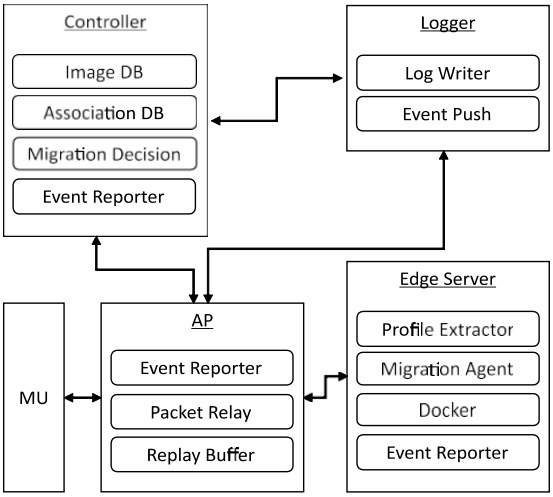
`LR` is a state reproduction method which is different from`FC` and `DC`. Instead of receiving state from ES(src), ES(tgt) collects the command trace executed at ES(src), starts the base image from scratch, and replays the commands locally to reconstruct the consistent state. The overall procedure of `LR` migration is illustrated in Fig. 5.



Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

The proposed replay buffer plays a key role in capturing thelast-minute state changes that are not included in what hasbeen transmitted from ES(src) to ES(tgt). The container onES(tgt) notifies AP-2 of its readiness only when the replaybuffer is empty and all queued commands are successfullyreplay on ES(tgt). In the meantime, the user's offloadingrequest packet received by AP-2 is relayed to ES(src) at AP-1.This packet relay and replay buffer scheme is particu- larly useful for container migration for the following tworeasons. The common stateful container migration approachis based on iterative transmission of memory dumps until apre-defined number of iterations is reached [6]. If there areremaining dumps to transmit after the iteration limit, it mayfail to recover the exactly same container at ES(tgt). Also,during migration the amount of last-minute state changes(i.e., dirty pages) to transmit can be larger than the com-mand itself which caused the changes. If so, transmitting thecommand log can be more efficient than sending the dirtypages in terms of bandwidth and time. The other is related tothe service availability and the fail-safe property. In general,common container migration techniques stop the container onES(src), and then, resume its replica on ES(tgt) at some pointsof migration procedure. However, such approach incurs ser-vice downtime no matter how short the period is. Also, if,for some reasons, it fails to launch the replica containeron ES(tgt) after stopping the original container on ES(src),the containerized service becomes unavailable, violating QoS requirement.

The proposed migration techniques, on the other hand, can effectively overcome the two problems. The container onES(src) keeps providing service until its replica on ES(tgt) becomes ready and fully-functioning, and as a result, there is no service-unavailable period. Also, if ES(tgt) fails to launcha container replica, the original one on ES(src) can continuethe service.



C. USE CASES AND APPLICATION PROPERTIES

The proposed three migration techniques have different char-acteristics and different use cases. Choosing an efficient migration technique in terms of migration time and networkload depends on both how each migration technique worksand the properties of the containerized service to be migrated.FC is a send-all migration and generates a large amount of network traffic since it transfers the base image(i.e., read-only layers) as well. Despite of its seemingly inef-ficiency, it is the only working solution when the base imageof the to-be-migrated container is available only at ES(tgt). If it is not the case, however, FC is always inferior to DC in terms of bandwidth use and migration time.

DC is particularly useful when state reproduction takes example, as reported in [42], training an Inception3 deep learning model for a plant leaf disease detection appli-cation took approximately 2 hours. However, the resulting model is only about 90MB in size. In such cases, as long as the network delay between ES(src) and ES(tgt) is small andbandwidth is large enough, transferring the trained model, i.e., DC, is more time and bandwidth efficient than receivingthe training data set and training the model at the ES(tgt), i.e., LR.

On the other hand, LR outperforms the rest if the staterestoration is faster than transferring the state. For example, augmentation [43] is a common trick to multiply the data setin deep learning for image-related applications to enhance the generalization performance of a trained model. It can be implemented with only a few lines of code (or command). The augmentation techniques are based on simple linear oper- ations, such as rotation, shift, shearing, zooming and flipping, and can be quickly done, while resulting in an additional voluminous data set. In such cases, instead of transferringall augmented data (i.e., DC), replaying the augmentation onES(tgt) can save time and bandwidth.

IV. PROPOSED OPTIMAL MIGRATION SYSTEM

This section introduces an optimal migration decision algo- rithm that chooses the best migration technique with respect to both the migration time and the network load. To do so, the algorithms considers characteristics of migration techniques and the application to be migrated together. Also, this section proposes a system design that can perform optimal migration autonomously.

A. PROPOSED SYSTEM DESIGN: OVERALL ARCHITECTURE The design of the proposed autonomous system that can perform optimal live migration is depicted in Fig. 6, and one possible deployment scenario is shown in Fig. 7. Note that a resourceful AP may include the ES inside, and both Controller and Logger can be implemented in a single machine. The proposed system design consists of the follow-ing modules:

FIGURE 6. The proposed autonomous optimal container migrationsystem design consists of controller, logger, AP, and edge server.

- Controller: When a handoff event is reported by Logger, Controller determines the optimal migration technique

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

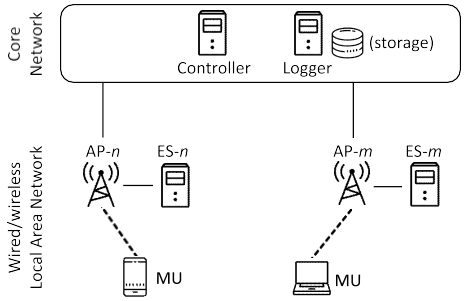


FIGURE 7. An illustrative deployment scenario of the proposed system.

and notifies both ES(src) and ES(tgt) of the decision as well as some extra information required to perform the chosen migration.

- Logger: It collects all event logs, and saves them to its local or remote storage. When the user handoff event is reported by an AP, Logger pushes the event to Controller.
- ES: It carries out containerized services and migration procedure.
- AP: In addition to providing the network access toassociated users, AP is responsible for packet relay andreplay buffer.
- Mobile User (MU) or user: MU continuously offloads tasks to containers, and due to its mobility, it incurs handoff between nearby APs.

The components inside each module are introduced below byusing the example scenario (Fig. 2) introduced in Section III. Suppose a user joins the network by associating with AP-1. The MU also initiates a containerized offloadingservice at ES-1, and AP-1 sends all event logs to Loggerby Event Reporter. Logger receives the logs and stores themby Log Writer. Among the received logs, certain events arepushed to Controller by the Event Push component. Forexample, user's association with AP-1 and the name of thebase image used for the containerized service are the eventsto be pushed to Controller. Then, Controller saves the AP ID(e.g., SSID, IP address and MAC address) and the image ID(e.g., image name and tag) in Association DB and Image DB, respectively.

In this paper, it is assumed that when a user triggers a handoff event, migration always executes. This is a reason- able assumption especially when the coverage of AP is large and user devices are moving at slow speed. In the case of IEEE 802.11, during the handoff procedure, a MU sends REASSOCIATION REQUEST to the new AP (i.e., AP-2 in Fig. 2), and this is the earliest moment a handoff event is detected except the AUTHENTICATION message exchange. The re-association event log is also sent to the Logger, and the Event Push component in Logger passes the handoff event toController.

When Controller is notified of the handoff event,the Migration Decision component starts by requesting information, called profile, to ES-1 that is needed to choosean optimal migration. ES-1's Profile Extractor gathers pro- file and sends it back to Controller. The set of information included in a profile is introduced in the following subsection. Also, Controller sends the AP-1 ID to AP-2, and AP-2 ID to AP-1. Upon receiving the profile, Controller determines the optimal migration technique, and sends the decision to both AP-1 and AP-2. Migration Agent at ES responds to the decision, and begins container migration. When AP-1 finishes state of log transmission, it sends a container start request to AP-2. AP-2's Migration Agent then allows Docker engine to start the container and carries out any remaining migration procedures, if any, before replaying commands inthe replay buffer. Once the migration procedure finishes, thecommands in the replay buffer is played on the container onES-2 until the buffer becomes empty. When there is no morecommands to replay, ES-2 notifies AP-2 of its availability, which is then passed to AP-1 so that it can stop its containerat ES-1.

B. PROPOSED OPTIMAL MIGRATION DECISIONALGORITHM

The proposed optimal migration decision algorithm jointly considers the migration time and the traffic load to be injected to the network as a result of migration. The migration time isthe time interval from when the migration is started at ES(src) to the moment the migrated container at ES(tgt) is readyto provide offloading service. The migration time affects service delay because when migration is in progress, a user request received by AP-2 is relayed to ES(src) at AP-1. Also, shortening the migration time saves computing resource on AP-1, since the container on ES(src) can stop and release the container resources only when the migration is finished.

Note that in this work, it is assumed that QoS is defined as a function of the service delay. However, there is no predetermined delay upper bound in this work since it can vary depending on application, domain, physical network performance, etc. Thus, in the remainder of this paper, it is presumed that a lower service delay is preferred. Also, the raw service delay will be measured and used for comparisonin Section VI. Yet, one QoS constraint we force in this studyis that an infinite delay is not permitted, which happens whenthere is a service outage.

Which migration to execute affects the network load or congestion since different migration technique injects differ- ent amount of traffic to the network. No matter how short the routing path from ES(src) to ES(tgt) is, the network traffic as a result of migration enters the core network (see Fig. 1). Thus, if the chosen migration technique generates too much network traffic, it may result in network congestion. Also, having more bits to transfer may increase the probabilityof transmission errors. Such errors incurs re-transmissions, increasing the migration time.

To formulate the optimal migration decision problem, the first step is to express the expected migration time of each migration technique. Let T_{FC} be the migration timefor FC, which is defined by the sum of the following

terms: time to generate an image t_{imggen}, time to generate a checkpoint t_{chkgen}, time to transmit the container image t_{imgtx},time to transmit the checkpoint t_{chktx}, time to start the con- tainer t_{ctst}, time to replay the buffered commands during migration t_{bufrep}. Let T_{DC} be the migration time for DC, whichis defined by the sum of the following terms: time to gener-as shown in Table 4 and 5. Therefore, the optimal migrationdecision among three techniques can be reduced to choosingthe best migration between DC and LR. The corresponding optimal migration decision problem is given below (denotedby P. 4).

ate a checkpoint t_{chkgen}, time to transmit the writable layer contents t_{difftx}, time to transmit the checkpoint t_{chktx}, time to start the container t_{ctst}, time to replay the buffered commands

$$f(x) = \frac{1}{\alpha}\{x \cdot T_{\mathrm{DC}} + (1-x)T_{\mathrm{LR}}\} + \frac{1}{\beta}\{x \cdot L_{\mathrm{DC}} + (1-x)L_{\mathrm{LR}}\} \quad (4a)$$

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

during migration t_{bufrep}. Let T_{LR} be the migration time for LR, which is defined by the sum of the following terms: time to generate a command trace t_{trgen}, time to transmit the trace t_{trtx}, time to start the container t_{ctst}, time to replay the received trace t_{rep}, time to replay the buffered commands during migration t_{bufrep}.

Given that the amount of user commands executed at a container affects the amount of memory state changes, it is assumed that t_{chkgen} t_{trgen}. Then, C can be defined as the common time factor included in T_{FC}, T_{DC} and T_{LR}. To be specific, $C = t_{chkgen} + t_{ctst} + t_{bufrep}$ for T_{FC} and T_{DC}, while $C = t_{trgen} + t_{ctst} + t_{bufrep}$ for T_{LR}. Finally, a shortened migration time expressions can be given as follows:

$$T_{FC} = t_{imggen} + t_{imgtx} + t_{chktx} + C, \quad (1)$$

$$T_{DC} = t_{difftx} + t_{chktx} + C, \quad (2)$$

$$T_{LR} = t_{trtx} + t_{rep} + C. \quad (3)$$

Note that the terms mentioned here are in the unit of seconds. Transmission times are the functions of both the number of bits to transfer and the bandwidth B (bits per second), where the latter is assumed to be known. The value of C to be used for evaluation in Section VI is acquired by the average of multiple experiments.

The amount of network traffic to be generated by three migration techniques is defined as follows. Let L_{FC} be the network load to be generated as a result of executing FC, and it is defined by the sum of the following terms: amount of data for transmitting an image l_{imgtx} and amount of data for transmitting a checkpoint l_{chktx}. Let L_{DC} be the network load to be generated as a result of executing DC, and it is defined by the sum of the following terms: amount of data for transmitting an writable layer l_{difftx} and amount of data for transmitting a checkpoint l_{chktx}. Let L_{LR} be the network load to be generated as a result of executing LR, and it is the amount of data for transmitting a command trace l_{trtx}. The terms mentioned here are in the unit of bits.

From the above equations for migration time and network load, it is obvious that DC is always superior to FC. This is because the complete container image, i.e., read-only layers and a writable layer, includes the writable layer, and in many cases, the size of the complete image is much larger than that of the writable layer. Therefore, in terms of both migration time and network traffic to be generated, DC always outper- forms FC. This claim also accords closely with the evaluation results to be introduced in Section VI-A. For example, the migration time of FC is 10-20 times larger than that of DC subject to: $x \in \{0, 1\}$, (4b)

where T_{DC} and T_{LR} are defined in Eq. 2 and Eq. 3, respec- tively, w + is a non-negative design parameter indi-cating the weight to the network load, t_{difftx} l_{difftx}/B, t_{chktx} l_{chktx}/B, t_{trtx} l_{trtx}/B $\in$ A R w 0, the migration technique minimizing the migration time is chosen as optimal solution, while as w gets larger, minimizing the network traffic becomes more important. The bandwidth B can be estimated by using well-known techniques such as packet pair probing [44]. The time-related terms, i.e., T_{DC} and T_{LR}, are measured in seconds, while network load-related terms, i.e., L_{DC} and L_{LR}, are in bits. This unit discrepancy may result in the problem of one having much large value than the other by nature. To make different terms have the same scale, both α and β are introduced to normalize the corresponding terms in the objective function, (4a).

The values needed to solve P. 4 are given, calculated or estimated right before solving the problem without actually performing either DC or LR. Some of the values are derived by the averaged values acquired from multiple evaluations. For example, the approach used for evaluation (Section VI) is to carry out multiple runs of experiments in advance to obtain the averaged values of t_{chkgen}, t_{ctst}, t_{trgen} and t_{imggen} for each scenario profile. On the other hand, other values can be calculated as follows. The size of the command trace can be given by parsing the log file as mentioned in Section III. The size of the writable layer can be calculated by the `du -h -apparent-size` command running at the `UpperDir` as aforementioned in Section III. The size of the checkpoint can be calculated by creating a checkpoint, which can be quickly done. The approximate size can also be found by using `docker stats` command which shows the real-time memory usage of containers. The size of the image that consists of read-only layers is given by checking the image size, and B can also be estimated with high precision as mentioned earlier. The only value that cannot be obtained when solving P. 4 is t_{bufrep} since it is hard to know which user actions might be performed during migration in advance. However, as it can be seen in Section VI, in some scenario profiles, DC and LR migration can quickly be done, and thus, it has a negligible effect. Most importantly, t_{bufrep} is part of the common constant time factor C and applied to both DC and LR. Such constant will be ignored when searching for an optimal solution.

problems are in general intractable, due to the small size of P. 4 that has only two binary decision variables, the optimal solution can be quickly found, for example by using the branch and bound algorithm [45]. However, instead of com-puting the optimum for P. 4 by using conventional solution methods, the proposed algorithm evaluates the problem witheach possible value of x separately, and chooses the best thatresults in the smallest objective value. The decision variable x takes value only between 0 and 1, and the terms included in $f(x)$ are linear, and thus, the proposed brute-force algorithmcan quickly terminate.

V. METHODOLOGY

This section explains the implementation detail, the set of scenarios used for evaluation, and the method of validation, performance measurement and analysis.

A. IMPLEMENTATION

For validation and performance evaluation, the proposed three migration methods and automated optimal migration system (Fig. 6) has been implemented, where the Python 3.8has been used to implement the programs for user, Controller, Logger, Edge Server, and AP. The testbed consists of off- the-shelf three Desktop PCs and a laptop: PC #1 for Con- troller and Logger, PC #2 for both AP-1 and ES-1, PC #3 for both AP-2 and ES-2, and the laptop for a MU. Forboth wired and wireless connectivity, a WiFi router (ipTIME A8004NS-M) has been used. The four machines are withina single local area network, where PCs and a laptop are connected via Ethernet cables and IEEE 802.11ac wireless channel, respectively. PCs are homogeneous with the follow- ing specifications: Intel Core(TM) i5-8500 CPU, 16GB RAM, 128GB SSD and 1000Mbps network interface card (NIC). The laptop is equipped with Intel Core(TM) i5-1035G4 CPU, 8GB RAM, 256GB SSD, and IEEE 802.11ac-compatible NIC. PCs are installed with Ubuntu 18.04.5 LTS oper- ating system, while Windows 10 is installed on the lap-top. The particular Docker and CRIU version used for implementation is 17.03.2-ce (build f5ec1e2) and 3.15, respectively.

In the testbed network, the background traffic has been controlled to be as little as possible so that it does not affect the delay and transmission rate during experiment. On aver-age, the round trip time (RTT) measured by a series of ICMPmessage exchanges between any two entities was approxi- mately 20 ms. To control both the bandwidth and latencyof APs, `tc` (i.e., a Linux tool to configure Linux Traffic Control) [46], has been used. With `tc`, the one-way net- work delay between two APs has been configured tobe 100 ms, which increases the service delay when themigration is in progress for the proposed packet relay. In this work, to precisely schedule handover events, sim- plified APs are implemented, and the user movements are emulated so that handover occurs at a scheduled moment in time.*SCENARIO*

Each user exclusively accesses its dedicated and isolated container for offloading service. As illustrated in the example scenario (Fig. 2) in Section III, MU associates with AP-1 andoffloads tasks to ES-1(src), at the beginning. As MU moves towards AP-2, handoff occurs, and the containerized serviceis migrated to ES-2(tgt).

To evaluate the proposed method on practical and con- trolled scenarios, the followings have been designed and implemented: two application services, service 1 (SVC-1) and service 2 (SVC-2), and two sequence of user actions (or commands) for task offloading, ACT-1 and ACT-2, that are applicable to both services. SVC-1 is a simple, light-weight application container that can startup fast and generates a small amount of data to be stored on the writable layer. On the other hand, SVC-2 is a relatively heavy application containerthat takes longer to startup, and user requests can generatea large amount of data to store. ACT-1 consists of actions that do not take long to complete. However, ACT-2 is a set of actions that takes longer than that of ACT-1. In addi- tion, the network bandwidth has been configured with two different configurations, namely, small bandwidth and large bandwidth. With the two applications, two action sequences, and two network bandwidth setups, eight scenarios or profiles are generated as summarized in Table 2.

TABLE 2. Eight different scenario profiles used for evaluation. Each profile/scenario is characterized by which service to use, which actionsequence to execute, and which network bandwidth configuration toapply for evaluation.

Scenario Label	Service Type	Action Sequence	Network
AC-LF/NC-L	SVC-1	ACT-1	Low bandwidth
AC-LS/NC-L	SVC-1	ACT-2	Low bandwidth
AC-HF/NC-L	SVC-2	ACT-1	Low bandwidth
AC-HS/NC-L	SVC-2	ACT-2	Low bandwidth
AC-LF/NC-H	SVC-1	ACT-1	High bandwidth
AC-LS/NC-H	SVC-1	ACT-2	High bandwidth
AC-HF/NC-H	SVC-2	ACT-1	High bandwidth
AC-HS/NC-H	SVC-2	ACT-2	High bandwidth

The combination of a service type, action sequence, and network configuration constitutes a scenario profile, labeled by AC-*st*/NC-*b* as shown in Table 2. By quantitatively ana- lyzing the effect of each action sequence on each applica- tion service with multiple experiments, it is found that each scenario generates a certain amount of data to be written to memory and writable layer, and also causes a certain periodof time for state reproduction on average. Based on such findings, an AC-*st* label is attached to each scenario, whereas NC-*b* simply indicates the size of the bandwidth configured. The detail explanation on the labels are given below, and theconfiguration details are listed in Table 3.

- $s \in \{L, H\}$ (Low, High): the total sum of both the amount of data to be stored at the writable layer and the size ofthe checkpoint (for `DC`),
- $t \in \{F, S\}$ (Fast, Slow): the state reproduction time for `LR` (note: this is different from replaying the commandsbuffered during packet relay), and

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

TABLE 3. Application and network configurations for evaluation: four configurations for the containerized application and two configurations for the network bandwidth make eight different scenario profiles in total, where WL is short for writable layer.

Application Configuration			Network Config.	
ID	Memory use + WL size (MB)	Trace replay time (second)	ID	Bandwidth (MB/s)
AC-LF	50	5	NC-L	10
AC-HF	200	5	NC-H	50
AC-LS	50	10		
AC-HS	200	10		

· $b \in \{L, H\}$ (Low, High): the size of the configured network bandwidth.

The configuration label *s* and *t* are related to `DC` and `LR`, respectively, while the label *b* affects the performance of all three migration techniques. In the case of `FC`, the amount of data to transfer is the size of the complete container image (i.e., read-only layers and a writable layer) and the check- point, and the former is not mentioned above. As discussed in Section IV, `FC` is always outperformed by `DC`, and thus, this work mainly focus on the performance comparison between `DC` and `LR`. The base image used for evaluation is the offi-cial `python:3.8` at DockerHub, and it is approximately 909MB in size.

For each scenario, each evaluation run lasts for 200 or 100 seconds, and MU generates a service request at the default rate, i.e., 1 request/second. MU handoffs from AP-1 to AP-2 after 50 seconds from the beginning or when sending 50th request to its associated AP.

B. PERFORMANCE MEASUREMENT

The key performance metrics considered in this study are the service delay, the migration time, the amount of traffic to be injected to the network as a result of migration, and the time taken to reproduce the given state. Since the timestamped logs that are related to service delay and migration time are sent to Logger, both are measured by inspecting the accumulated logs which act as regarded as raw data in this study. The timestamp included in each log does not represent when it is received by Logger, but when the event actually occurred. On the other hand, The amount of traffic and the state repro-duction time have been computed in advance.

To precisely measure the service delay before/during/after migration, a MU is configured to generate offloading ser- vice request at a fixed interval, i.e., 1 REQ/s. A user's offloading request is forwarded by an associated AP to the co-located ES, where the request is actually processed. Once the ES completes processing the offloaded task, it sends an application-layer positive acknowledge (ACK) back to the user via the AP. Thus, the response time or service delay can be measured by the elapsed time between the moment a user sends an offloading request and the moment the user receives the corresponding ACK. The corresponding logs are transmitted to Logger, and by identifying both logs, the migration time can be calculated. The migration time is the elapsed time between the moment the migration is started at ES-1(src) and the moment the migrated container at ES-2(tgt) is ready to provide the offloading service. The corresponding logs are transmitted to Logger, and by which, the migration time can be calculated. There are predefined scenario profiles and they assumed to be the same throughout the experiments. Thus, for a scenario profile, the number of bits to be transmitted as well as the state reproduction time remains the same. Multiple experiments have been carried out and then the average has been taken in advance to calculate both.

Note that the task offloading, in general, can reduce the energy consumption of an end device, but the direct relation-ship between the proposed migration and the saved power consumption is beyond the scope of this study. In a nutshell, as studied in [47], [48], the power consumption of a user device is proportional to the computation load or the number of CPU cycles required to process a given job. Since an offloaded job is processed not on user device, but on edge server, edge computing can effectively achieve power saving of end devices.

C. VALIDATION

The validation of the proposed optimal migration has been carried out from various aspects based on the accumulated logs at Logger. The occurrence of migration has been vali- dated by inspecting the logs indicating which ES processed the offloading requests from the user. Whether the ES car- ried out the chosen, optimal migration has been validated by inspecting the log from Controller regarding its decision and the logs from both ES-1(src) and ES-2(tgt) regrading the migration they actually performed. Each task offload- ing request from a user is sequentially numbered, and the corresponding ACK from the serving ES includes the same number. Thus, whether or not there has been any offloading

service outage has been validated by inspecting the numbersof the request-ACK pairs. If there are N number of requests and the corresponding ACKs in the logs without anything missing, it is regarded that there was no offloading service outage.

D. RESULTS ANALYSIS

For each scenario profile, five runs of experiments have beencarried out. To minimize the effect of the outliers and to draw the common behavior on each scenario, an average has been taken out of the results from multiple experiments. Such averaged results are, then, used for performance comparisonand analysis.

VI. EXPERIMENTAL EVALUATION

This sections shows the performance evaluation results of the proposed optimal migration running on the testbed with the scenarios configurations that are introduced in Section V.Note that for all experiments carried out and report in this section, there was no service outage, which is validated by checking any missing requests or ACKs in the log at Logger.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

IN-DEPTH INSPECTION ON THE BEHAVIORS OF THETHREE MIGRATION TECHNIQUES

1) MIGRATION ON AC-LF/NC-L SCENARIO

To begin with, AC-LF/NC-L and AC-LF/NC-H scenarios have been carried out to take a close look at how each migration technique behaves and how the per-request servicedelay (i.e., response time) varies. Both AC-LF/NC-L and AC- LF/NC-H have the same application configuration. However,the network bandwidth is different, which affects the elapsedtime to transmit the migration-related data, resulting in dif- ferent migration time. Fig. 8 and Fig. 9 show the experi- ment result for the AC-LF/NC-L scenario profile that lasted for 200 seconds. To be specific, Fig. 8 shows the per-requestresponse time trace for FC (Fig. 8(a)), DC (Fig. 8(b)) and LR (Fig. 8(c)). Fig. 9 shows the on average per-request response time for three migrations together.

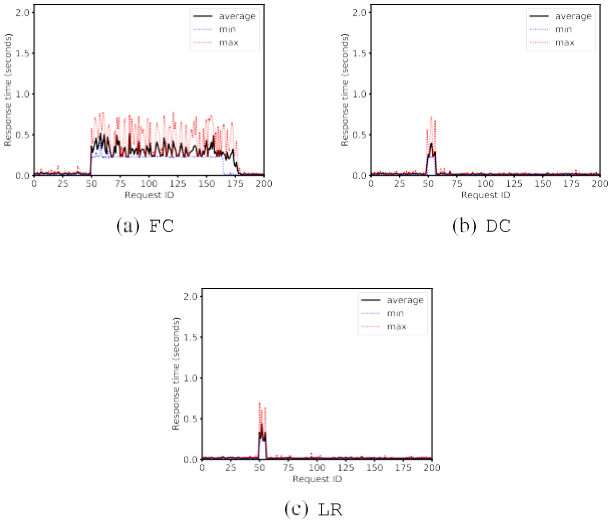


FIGURE 8. Dark solid line shows the average per-request response timetrace on the AC-LF/NC-L scenario. The migration is triggered when the 50th request is sent out from the MU. The red and blue dotted line represents the largest and shortest response time for each request, respectively.

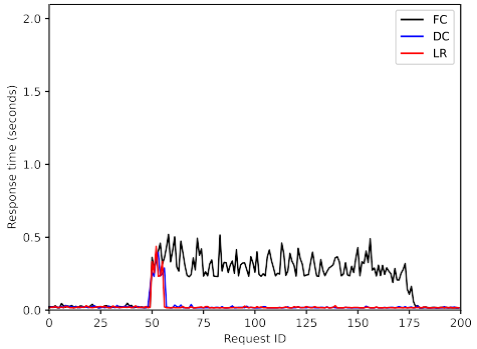


FIGURE 9. Average per-request response time trace for three migrationson the AC-LF/NC-L scenario.

As expected, FC spent the most time on migration, result-ing in the largest response time among the three migration

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

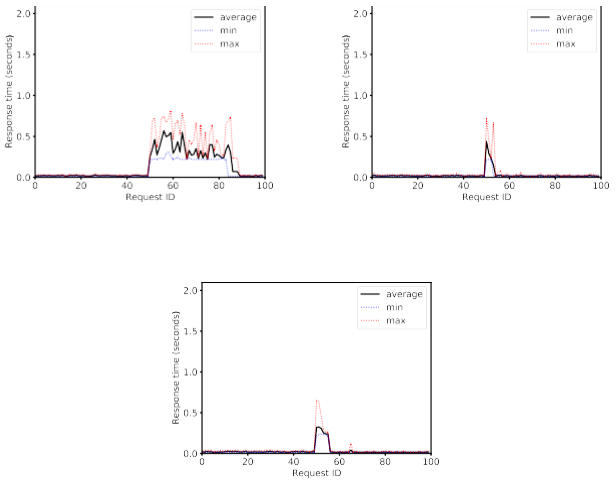
TABLE 4. Average migration time and response time for three migrationson the AC-LF/NC-L scenario.

Average (second)	FC	DC	LR
Migration time	124.9040	6.9080	5.9728
Per-request response time	0.2029	0.0289	0.0266

techniques. The response time is proportional to the migrationtime, because the packet relays occur between AP-1 and AP-2 during migration, and the RTT between the two is relativelylarge. While migration is in progress, the MU that has associated with AP-2 communicates with AP-2. However, itsfunctioning container is still in ES-1 at AP-1, and the onethat is co-located with AP-2 has not been started yet. Thus,AP-2 has to relay the MU's requests to AP-1 so that the MU'scontainer in ES-1 can respond to the requests. Due to thisrelaying, the response time becomes larger during migration.In this work, ES(src) with `FC` is implemented to perform the following tasks in sequence for migration: T1) generatesa complete container image, including both the read-onlylayer and the writable layer, T2) transmits the image, T3)creates a checkpoint, and 4) transmits the checkpoint.In the testbed setup, T1 takes a few seconds due to thelarge size of the image. Also, for the low bandwidth in theAC-LF/NC-L scenario profile, it took approximately 100 sec-onds long to transmit the image. This is because the entiretraffic `FC` generates is approximately 960MB (i.e., 909MBfor the read-only image and 50MB for both checkpoint andwritable layer contents). Upon receiving the complete con-tainer image and the checkpoint, ES-2 performs the followingtasks: T1) loads the received image, T2) creates a containerfrom the image, T3) starts the container with the receivedcheckpoint. The three tasks also requires a few seconds of time.

On the other hand, both `DC` and `LR` resulted in a muchshorter migration time than `FC`. The main reason is the reduced amount of data to transmit. `DC` spends approxi- mately 5 second in transmission to deliver 50MB of data which is the sum of the writable layer and checkpoint. `LR` transmits only a small-sized text, i.e., command log, which is approximately 1MB, and spends 5 seconds in replay- ing the received command trace. `LR` migration was slightly shorter than `DC`, although the difference is negligible. This ismainly because of the time taken to generate and restore froma checkpoint.

As it can be seen in the summary of migration time and response time on average shown in Table 4, the migration time directly affects the service delay or the response time for the user requests. As a result, `FC` resulted in the largest response time of 0.2029 seconds on average. The migration time of `DC` is slightly longer than that of `LR`, but, the average response time between the two does not much differ. This is because the additional delay incurred by `DC` compared to `LR` has become insignificant as the experiment progresses. If theexperiment lasts longer, the average delay between `DC` and `LR` will become much closer.



(a) FC (b) DC (c) LR

FIGURE 10. Dark solid line shows the average per-request response timetrace on the AC-LF/NC-H scenario. The migration is triggered when the 50th request is sent out from the MU. The red and blue dotted line represents the largest and shortest response time for each request, respectively.

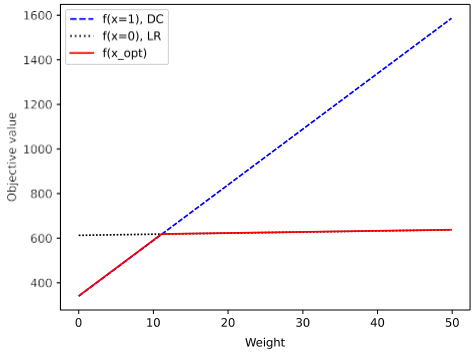


FIGURE 12. Objective value of P. 4 evaluated for the AC-LF/NC-H scenario. The blue dashed line and black dotted line show the objective value when $x = 1$ and $x = 0$, respectively, and the red solid line shows the optimal migration with respect to the weight w.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

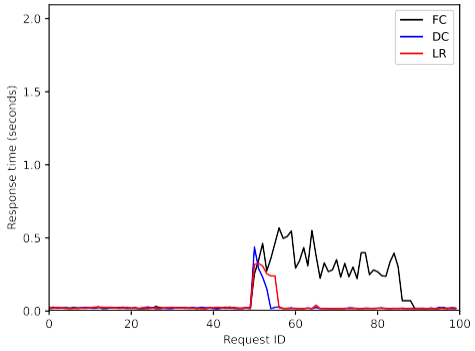


FIGURE 11. Average per-request response time trace for three migrationson the AC-LF/NC-H scenario.

TABLE 5. Average migration and response time for three migrations onthe AC-LF/NC-H scenario.

Average (second)	FC	DC	LR
Migration time	37.1064	3.4032	6.1295
Per-request response time	0.1390	0.0293	0.0351

From the optimal migration point of view, LR is always optimal for this scenario profile. LR always injects less traffic to the network. Moreover, for the AC-LF/NC-L configura- tion, LR resulted in a shorter migration time. Thus, LR is tobe chosen as optimal for any value of the weight parameter *w*. Note that the values of *alpha* and *beta* are configured to 0.01and 2, respectively, which are empirically obtained.

2) MIGRATION ON AC-LF/NC-H SCENARIO

Fig. 10, Fig. 11 and Table 5 show the evaluation resultsfor the AC-LF/NC-H scenario profile. The evaluation iscarried out for 100 seconds, not 200, and this affects the on average response time. It is worth noting thatmigration methods that transfer much data take advantage of the increased bandwidth. As a result, the migration time for both FC and DC decreased much, while that ofLR did not significantly change compared to the previous AC-LF/NC-L scenario profile. As a result, the migration time of FC reduced to 37.1064 seconds, which contributed to the reduced per-request response time. The increasedbandwidth also reduced the migration time of DC. How- ever, the migration time of LR is not significantly changed, because LR transmits only a little from ES-1 to ES-2. The only data to transmit is the command trace which is a text-only, small-sized file. The reduced migration time of DCmade the response time shorter than that of LR on average.

From the optimal migration point of view, an interesting result was found as shown in Fig. 12. Although LR injectsless traffic to the network, DC resulted in a shorter migration time for the AC-LF/NC-H configuration. Thus, in contrast to the previous AC-LF/NC-L scenario profile, LR is not always the optimal migration on AC-LF/NC-H. A small value of *w* tends to ignore the effect of the amount of network traffic togenerate, and thus chooses DC as an optimal solution. How- ever, when $w > 11.1277$ the importance on the migration time vanishes, and thus, LR is chosen to be optimal.

From the evaluation results on both AC-LF/NC-L and AC-LF/NC-H, it can be seen that the change in bandwidth affects the performance of both FC and DC that have relativelylarge traffic to transmit. On the other hand, the performance of LR that does not have much to transmit does not vary much for the bandwidth size.

A. PERFORMANCE COMPARISON BETWEEN DC AND LR MIGRATION

As discussed in Section IV, FC is always outperformed by DC.Thus, from now on, the performance comparison between DC and LR will be discussed, excluding FC. Also, the evaluation is carried out only for 100 seconds.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

1) MIGRATION OF SERVICE 1 CONTAINER

Both Fig. 13 and Table 6 show the evaluation results of DC and LR on the AC-LS/NC-L and AC-LS/NC-H scenario profiles. As it can be seen, the migration time of DC changed much

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

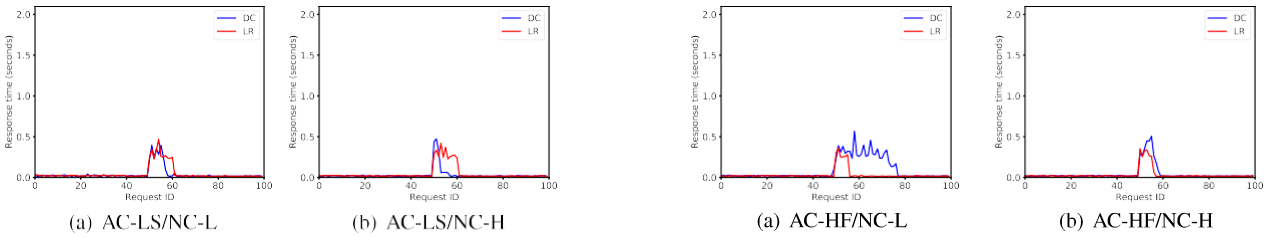


FIGURE 13. Average per-request response time trace for DC and LR on AC-LS/NC-L and AC-LS/NC-H scenario profiles.

TABLE 6. Average migration time and response time for DC and LR on AC-LS/NC-L and AC-LS/NC-H scenario profiles.

Profile	Average (second)	DC	LR
AC-LS/ NC-L	Migration time	7.4572	10.9406
	Per-request response time	0.0411	0.0488
AC-LS/ NC-H	Migration time	3.7277	10.8831
	Per-request response time	0.0324	0.0487

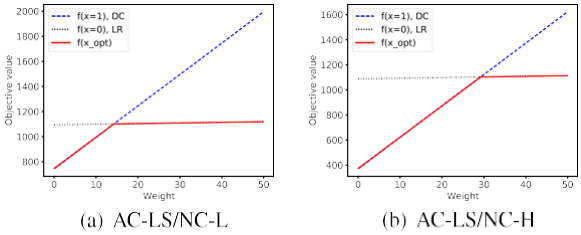


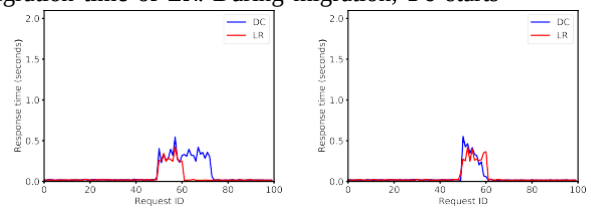
FIGURE 14. Objective value of P. 4 evaluated for AC-LS/NC-L and AC-LS/NC-H scenario profiles. The blue dashed line and black dotted line show the objective value when x 1 and x 0, respectively, and the red solid line shows the optimal migration with respect to the weight w.

between the two scenario profiles for the network bandwidth difference, but it was not the case to LR. With the ACT-2 action sequence, the ES-2 spends approximately 10 seconds only to replay the commands in the log. On the other hand, DC only needed to transmit the writable layer and checkpoint which amount to 50 MB in total, and as a result, the migration time is smaller than that of LR.

Fig. 14 shows the change of the objective value of P. 4 evaluated for AC-LS/NC-L and AC-LS/NC-H scenario pro-files. From the perspective of migration time, LR is outper- formed by DC, and thus, the optimal migration changes as w increases. For AC-LS/NC-L and AC-LS/NC-H scenario profiles, the optimal solution changes from DC to LR when w 14.2179 and w 29.2057, respectively. On the AC- LS/NC-H scenario profile, a larger value of w is required to switch the optimal solution from DC to LR compared to the AC-LS/NC-L scenario profile. This is because the migration time of DC in AC-LS/NC-H is less than that in AC-LS/NC-L, while LR results in almost identical migration time in both scenario profiles.

2) MIGRATION OF SERVICE 2 CONTAINER

The startup procedure of the SVC-2 application container from scratch is more complex than that of SVC-1, and it affects the migration time of LR. During migration, DC starts



Industrial Engineering Journal
ISSN: 0970-2555
Volume : 51, Issue 04, April : 2022

(c) AC-HS/NC-L (d) AC-HS/NC-H

FIGURE 15. Average per-request response time trace for DC and LR on AC-HF/NC-L, AC-HF/NC-H, AC-HS/NC-L and AC-HS/NC-H scenario profiles.

TABLE 7. Average migration time and response time for DC and LR on AC-HF/NC-L, AC-HF/NC-H, AC-HS/NC-L and AC-HS/NC-H scenario profiles.

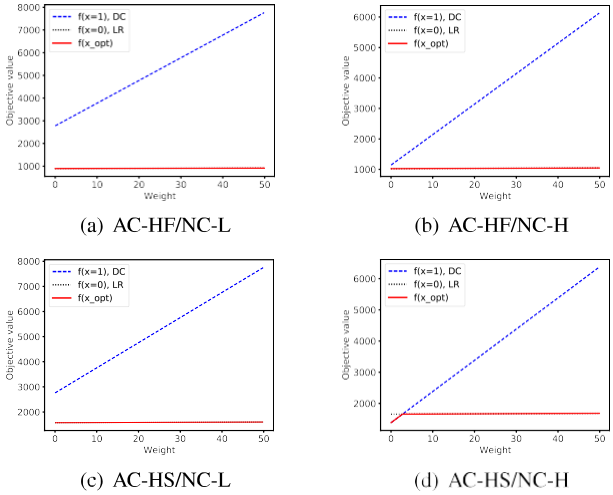
Profile	Average (second)	DC	LR
AC-HF/ NC-L	Migration time	27.7610	8.9276
	Per-request response time	0.0943	0.0341
AC-HF/ NC-H	Migration time	11.4055	10.2165
	Per-request response time	0.0457	0.0357
AC-HS/ NC-L	Migration time	27.5847	15.7138
	Per-request response time	0.0913	0.0472
AC-HS/ NC-H	Migration time	13.8344	16.5808
	Per-request response time	0.0505	0.0509

a container at ES-2 from the most-recent state, and it does not start a container from scratch. On the other hand, LR starts a container from scratch on ES-2, and then, replays the received the trace log. The time-consuming startup of SVC-2 delays the entire LR migration process, which is noticeable in the experiment results.

The evaluation with the container running SVC-2 has beencarried out on different action sequence and network band- width configurations, and the results are shown in Fig. 15 and Table 7. SVC-2 containers result in a larger checkpoint and writable layer than SVC-1 containers, and the amountof network traffic to generate by DC is increased to 200 MB. This is why DC took much longer for migration than LR whenthe network bandwidth is small–see Fig. 15(a) and Fig. 15(c).On the other hand, when the network bandwidth is large, the migration time for DC is close to or shorter than that of LR by reducing the data transmission time.

The migration time of DC largely depends on the network bandwidth, and thus, DC resulted in a shorter migration time when the bandwidth is larger, i.e., Fig. 15(b) and Fig. 15(d).On the other hand, the migration time of LR is largely affectedby the time taken to replay the transmitted log. Thus, LR resulted in a shorter migration time only when the replay time is shorter, i.e., Fig. 15(a) and Fig. 15(b).

Out of the four scenario profiles, DC outperforms LR only on the AC-HS/NC-H scenario profile with respect to the



(a) AC-HF/NC-L (b) AC-HF/NC-H

(c) AC-HS/NC-L (d) AC-HS/NC-H

FIGURE 16. Objective value of P. 4 evaluated for AC-HF/NC-L, AC-HF/NC-H, AC-HS/NC-L and AC-HS/NC-H scenario profiles. The blue dashed line and black dotted line show the objective value when x 1and x 0, respectively, and the red solid line shows the optimal migration with respect to the weight w.

migration time. Thus, for the other three scenario profiles, LR is chosen to be optimal as shown in Fig. 16. On the AC-HS/NC-H scenario profile, the migration time of DC is slightly less than that of LR. Thus, for a small value of w, DC ischosen to be optimal. However, for the value of $w > 2.7602$, LR is taken as an optimal migration technique.

VII. CONCLUSION

This paper has proposed three seamless, stateful migration techniques for containerized services. Both FC and DC arestate duplication methods in that they transfer state to the target edge server. On the other hand, LR is a state reproduc- tion method that replays the command trace to **build** a con- tainer with a consistent state. To capture the last-minute state changes, i.e., the state changes that are not included in what has been transmitted to the target edge server, this paper has proposed a packet relay and buffer replay method. Then, thispaper has proposed a system design for an autonomous opti-mal migration selection system. It chooses the optimal migra- tion technique considering the characteristics of the appli- cation to be migrated and the migration methods together. The proposed optimal migration selection problem considers both the migration time and the network load, and makes an optimal decision according to the tunable weight parameter. This paper has introduced implementation details on the three migration techniques as well as the autonomous migration system, and also carried out experiments. The results have revealed that the change in bandwidth largely affects the performance of DC. On the other hand, the performance of LR depends mainly on the application property of the trace replay time. As a result, it is found that both the characteristics of migration techniques and the properties of the applications to be migrated should be jointly considered when making a decision on optimal migration.

REFERENCES

[1] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang,''A survey on the edge computing for the Internet of Things,'' *IEEE Access*,vol. 6, pp. 6900–6919, 2017, doi: 10.1109/ACCESS.2017.2778504.

[2] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa,and I. Yaqoob, ''Big IoT data analytics: Architecture, opportunities, and open resource challenges,'' *IEEE Access*, vol. 5, pp. 5247–5261, 2017, doi:10.1109/ACCESS.2017.2689040.

[3] L. Zhao, J. Wang, J. Liu, and N. Kato, ''Optimal edge resource allocation in IoT-based smart cities,'' *IEEE Netw.*, vol. 33, no. 2, pp. 30–35, Mar. 2019, doi: 10.1109/MNET.2019.1800221.

[4] K. Kumar and Y.-H. Lu, ''Cloud computing for mobile users: Can offloading computation save energy?'' *Computer*, vol. 43, no. 4,pp. 51–56, Apr. 2010, doi: 10.1109/MC.2010.98.

[5] *AWS Auto Scaling*. Accessed: Sep. 24, 2021. [Online]. Available: https://aws.amazon.com/autoscaling/

[6] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito,''Container migration in the fog: A performance evaluation,'' *Sensors*, vol. 19, no. 7, pp. 1–22, Mar. 2019, doi: 10.3390/s19071488.

[7] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, ''Mobile edge computing: A survey,'' *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018, doi: 10.1109/JIOT.2017.2750180.

[8] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, ''Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges,'' *J. Netw. Comput. Appl.*, vol. 52, pp. 154–172,Jun. 2015, doi: 10.1016/j.jnca.2015.03.001.

[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, ''A survey on mobile edge computing: The communication perspective,'' *IEEE Com- mun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017, doi: 10.1109/COMST.2017.2745201.

[10] M. Satyanarayanan, ''The emergence of edge computing,'' *Computer*,vol. 50, no. 1, pp. 30–39, 2017, doi: 10.1109/MC.2017.9.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, ''Edge computing: Vision and challenges,'' *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[12] W. Shi and S. Dustdar, ''The promise of edge computing,'' *Computer*,vol. 49, no. 5, pp. 78–81, 2016, doi: 10.1109/MC.2016.145.

[13] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, ''Container-as-a-service atthe edge: Trade-off between energy efficiency and service availability at fog nano data centers,'' *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56,Jun. 2017, doi: 10.1109/MWC.2017.1600427.

[14] S. Wang, J. Xu, N. Zhang, and Y. Liu, ''A survey on service migration in mobile edge computing,'' *IEEE Access*, vol. 6, pp. 23511–23528, 2018, doi: 10.1109/ACCESS.2018.2828102.

[15] V. Prokhorenko and M. A. Babar, ''Architectural resilience in cloud, fog and edge systems: A survey,'' *IEEE Access*, vol. 8, pp. 28078–28095, 2020, doi: 10.1109/ACCESS.2020.2971007.

[16] M. Gusev and S. Dustdar, ''Going back to the roots—The evolution of edgecomputing, an IoT perspective,'' *IEEE Internet Comput.*, vol. 22, no. 2, pp. 5–15, Mar. 2018, doi: 10.1109/MIC.2018.022021657.

[17] *Docker*. Accessed: Aug. 5, 2021. [Online]. Available: https://www. docker.com/

[18] K. Govindaraj and A. Artemenko, ''Container live migration for latency critical industrial applications on edge computing,'' in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2018, pp. 83–90.

[19] P. Saha and A. Beltre, ''Evaluation of Docker containers for scientific workloads in the cloud,'' in *Proc. ACM PEARC*, Pittsburgh, PA, USA, 2018, pp. 1–8, doi: 10.1145/3219104.3229280.

[20] *Checkpoint/Restore in Userspace*. CRIU. Accessed: Aug. 11, 2021. [Online]. Available: https://criu.org/

[21] P. Karhula, J. Janak, and H. Schulzrinne, ''Checkpointing and migration of IoT edge functions,'' in *Proc. 2nd Int. Workshop Edge Syst., Analytics Netw. (EdgeSys)*, 2019, pp. 60–65, doi: 10.1145/3301418.3313947.

[22] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, ''Voyager: Complete con- tainer state migration,'' in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.(ICDCS)*, Jun. 2017, pp. 2137–2142, doi: 10.1109/ICDCS.2017.91.

[23] C. Dupont, R. Giaffreda, and L. Capra, ''Edge computing in IoT context: Horizontal and vertical Linux container migration,'' in *Proc.Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–4, doi: 10.1109/GIOTS.2017.8016218.

[24] *Kubernetes*. Accessed: Aug. 11, 2021. [Online]. Available: https:// kubernetes.io/

[25] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, ''Autonomic vertical elasticity of Docker containers with ELASTICDOCKER,'' in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 472–479.

[26] L. Ma, S. Yi, N. Carter, and Q. Li, ‘‘Efficient live migration of edge services leveraging container layered storage,’’ *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019, doi: 10.1109/ TMC.2018.2871842.

[27] L. Ma, S. Yi, and Q. Li, ‘‘Efficient service handoff across edge serversvia Docker container migration,’’ in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, Oct. 2017, pp. 1–13.

[28] C. Yu and F. Huan, ‘‘Live migration of Docker containers through loggingand replay,’’ in *Proc. 3rd Int. Conf. Mechatronics Ind. Informat.*, 2015, pp. 623–626.

[29] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, ‘‘Live migration of virtual machine based on full system trace and replay,’’ in *Proc. 18th ACM Int. Symp. High Perform. Distrib. Comput. (HPDC)*, 2009, pp. 101–110.