*sustainability*

MDPI

*Article*

# A Predictive Checkpoint Technique for Iterative Phase of Container Migration

**Gursharan Singh [1], Parminder Singh [1,2,*], Mustapha Hedabou [2], Mehedi Masud [3,*] and Sultan S. Alshamrani [4]**

[1] School of Computer Science and Engineering, Lovely Professional University, Phagwara 144401 India; gursharan.16967@lpu.co.in
[2] School of Computer Science, University Mohammed VI Polytechnic, Ben Guerir 43150, Morocco; mustapha.hedabou@um6p.ma
[3] Department of Computer Science, College of Computer and Information Technology, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia
[4] Department of Information Technology, College of Computer and Information Technology, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia; susamash@tu.edu.sa
* Correspondence: parminder.singh@um6p.ma (P.S.); mmasud@tu.edu.sa (M.M.)

**Abstract:** Cloud computing is a cost-effective method of delivering numerous services in Industry 4.0. The demand for dynamic cloud services is rising day by day and, because of this, data transit across the network is extensive. Virtualization is a significant component and the cloud servers might be physical or virtual. Containerized services are essential for reducing data transmission, cost, and time, among other things. Containers are lightweight virtual environments that share the host operating system's kernel. The majority of businesses are transitioning from virtual machines to containers. The major factor affecting the performance is the amount of data transfer over the network. It has a direct impact on the migration time, downtime and cost. In this article, we propose a predictive iterative-dump approach using long short-term memory (LSTM) to anticipate which memory pages will be moved, by limiting data transmission during the iterative phase. In each loop, the pages are shortlisted to be migrated to the destination host based on predictive analysis of memory alterations. Dirty pages will be predicted and discarded using a prediction technique based on the alteration rate. The results show that the suggested technique surpasses existing alternatives in overall migration time and amount of data transmitted. There was a 49.42% decrease in migration time and a 31.0446% reduction in the amount of data transferred during the iterative phase.

**Keywords:** container migration; iterative dump; memory prediction; dirty pages; LSTM



**Citation:** Singh, G.; Singh, P.; Hedabou, M.; Masud, M.; Alshamrani, S.S. A Predictive Checkpoint Technique for Iterative Phase of Container Migration. *Sustainability* **2022**, *14*, 6538. https://doi.org/10.3390/su14116538

Academic Editors: Jawad Ahmad, Muhammad Awais, Mohsin Raza and Ghufran Ahmed

Received: 6 April 2022
Accepted: 16 May 2022
Published: 26 May 2022



## 1. Introduction

Since software emulates components of a system, virtual machines have long been the primary means of delivering virtualization in the industrial Internet of Things (IIoT). Virtualization permits activities to be performed in isolated environments, allowing for greater consistency because an emulation abstracts the entire underlying system [1]. Containers have emerged as a viable alternative to virtual machines in the past few years. Containers existed previously, but they witnessed a significant surge in popularity when the container framework Docker was introduced in 2013. Docker introduced capabilities that allowed users to quickly construct, distribute, and build upon each other's containers, which helped their growth since users could utilize pre-existing containers.

Containers are frequently viewed as lightweight virtual computers with short boot times and low resource consumption [2]. One significant reason for this is that containers, unlike virtual machines, run on the host machine's kernel, as shown in Figure 1. This is advantageous in multi-tenant cloud providers and data centers since each bare-metal system is likely to run more instances, and instances may be launched or restarted more effectively, resulting in a much higher quality of service. Migration is a critical technique in

the context of virtual machines and containers [3]. The process of moving an instance of a container that is in a running state across hosts is known as migration [4]. Depending upon the nature of the task or according to the demands of the customer, a container migration can be live or non-live. While moving an instance, live migration means the user is unaware of this migration. The state of the container is migrated prior to the container migration [5]. This technology is critical in various virtualized settings because it allows instances of virtual machines or containers to be transferred across hosts while retaining state, enabling effective load balancing and more straightforward maintenance with minimum impact.

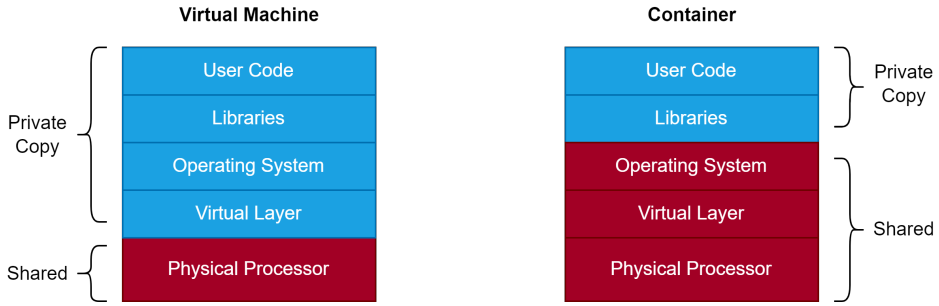


**Figure 1.** Difference in the architecture of virtual machine and container

### *1.1. Open Container Initiative*

Containers have been around since the chroot system function was added to the Linux kernel in 1979. By enabling programs to alter their apparent root directory, this system function facilitated the isolation of file system hierarchies.

Nowadays, there are several process isolation methods available for Industry 4.0. Many of them are made possible by a Linux kernel feature known as a namespace.A namespace enables the isolation of distinct aspects of a process within a namespace. It restricts the process's awareness to its current namespace or nested namespace. The available namespaces are Cgroup, IPC, Network, Mount, PID, Time, User, and UTS. Furthermore, the use of system resources by a process, such as CPU and memory, may be restricted and monitored using cgroups, a component of the Linux kernel. The concept of containers has been around for several decades; it was obscure until 2013 when Docker [6] received much attention. Docker is a framework for building, operating, and managing containers. When it was initially published, it had numerous essential container functionalities. The most notable is the layered picture approach it employs for container creation. This allows the construction of container images made up of layers that individually alter the underlying file system utilized by the container, and then these images can be shared according to the requirement.

As per the IIoT requirements, the Linux Foundation started a project with Docker in 2015 and this project was named the Open Container Initiative (OCI). The main purpose was to standardize the containers for global acceptance with a complete architecture. The runtime and its conforming containers include events of container lifecycle and activities. The container state collects the container id, the version of the specification, the file system path, and runtime data. The lifecycle specifies the events that occur during execution and the action taken in response to those events.

### *1.2. Migration Techniques*

The container gained popularity when Docker introduced its natural flexibility. It can run in the kernel of the host's operating system and be moved to another operating system without the bundle of system dependencies. It can be performed using the concept of namespaces as discussed in Section 1.1. Checkpoint and restore in userspace (CRIU) has provided a feature to make a checkpoint of a running container and migrate it to another host [7]. The stateless migrations are not concerned with moving any state across the hosts throughout the migrations, but stateful migrations are. As a result, stateless migration procedures are typically fairly straightforward, with the only steps necessary to conduct a stateless migration being to start a new container on the target server and then to delete the

container on the source host [8]. Because of the ease of stateless migrations, the migration mechanisms presented are all stateful.

Furthermore, these states are divided into the runtime state and permanent storage. The runtime state is generally composed of volatile data, such as CPU state, open file descriptors, and memory pages, which are lost when the container quits. Permanent storage generally comprises data not destroyed when the container is shut down, such as volumes mounted inside the container. The following migration strategies are primarily concerned with runtime states because shared volume approaches often manage permanent storage.

#### 1.2.1. Cold Migration

This method is likely the easiest, but it has the most significant disadvantage due to its lengthy downtime. Once the container dump is completed, the container is immediately terminated. The dumped state is then transferred to the destination host. The container will be restarted with the same state from the received dump as shown in Figure 2. The container is simply terminated, its state is dumped, and the destination host will receive this. The container is restored with the received dumped state at the destination host. As a result, there is a long overall migration time and downtime. This approach is frequently criticised, as it offers minimal benefits other than automating the migration process and transferring a small quantity of data. It is also known as offline migration.

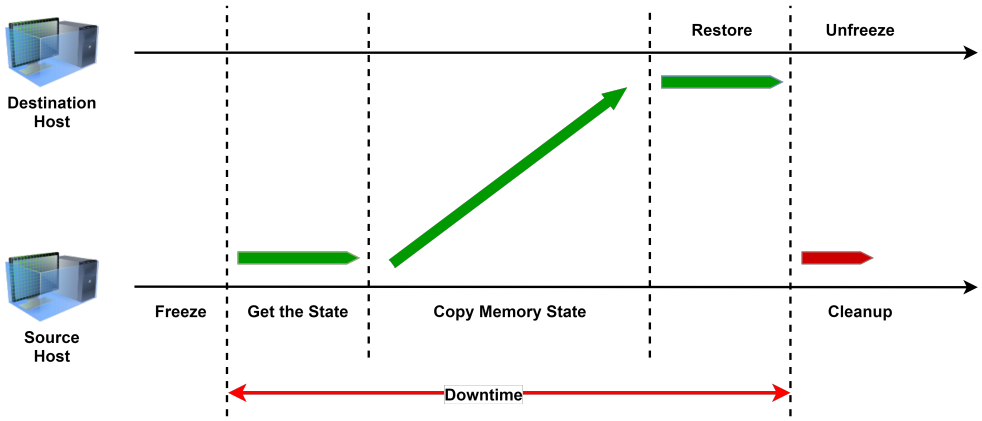


**Figure 2.** Phases of cold migration process of containers [9].

The main drawback of this cold migration is the long downtime. The services provided by the container will be stopped for a long period, which is unacceptable according to industry standards.

#### 1.2.2. Pre-Copy Migration

It is suitable for live migrations. While a container is running on the source host and it is decided to migrate a container to the destination host, the pre-dump phase will be initiated and commence the transferring state and the memory pages related to the container. During this process, the container is still running on the source host as shown in Figure 3. This pre-dumped state often contains only the container's memory pages, not the entire container's state. It may also contain additional running state data. The container state may be modified again while transferring the state in the previous round and marked as updated with dirty bits to avoid any conflict in the complete state of the container.

An iterative dump will be performed to send the incremental state. Iterations may vary depending upon the container dump size or threshold. When the container's iterative dump state is completed, the last transfer will be initiated and called a final dump. This includes all changes made after the last iterative dump and is used to set the state of the container to the latest state.

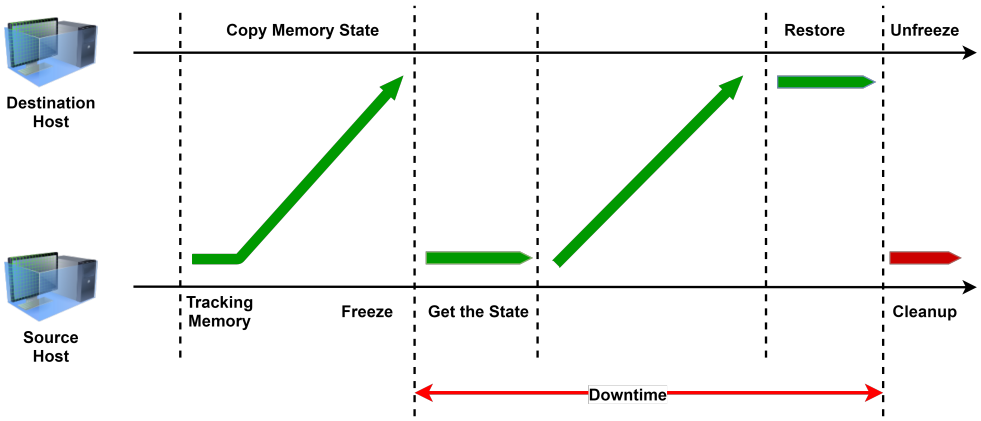


**Figure 3.** Phases of pre-copy container live migration [9].

In pre-copy, the downtime is less but the total migration time may be longer than in cold migration. If the data transfer during the iterative phase can be controlled so that re-transmission of pages is minimized, then this migration approach is best suited for container migration with low downtime.

#### 1.2.3. Post-Copy Migration

Just transferring the state and restoring the container would be the same as cold migration. It stops the container when a migration request is granted and then dumps its state to the destination host as shown in Figure 4. Instead of sending the complete state, it just sends the necessary execution state to start the container at the destination. When the container starts at the destination, the host then starts generating page faults according to the requirement. The source will handle the request for a demanded page, and the required page will be transferred to the destination. Once all the required pages are transferred, the dump copy can be removed from the source. There are some major drawbacks:

1. If the target host crashes in the process of demand paging, then the latest state of the container becomes unrecoverable.
2. If the source host crashes, then the container cannot access the remaining pages.

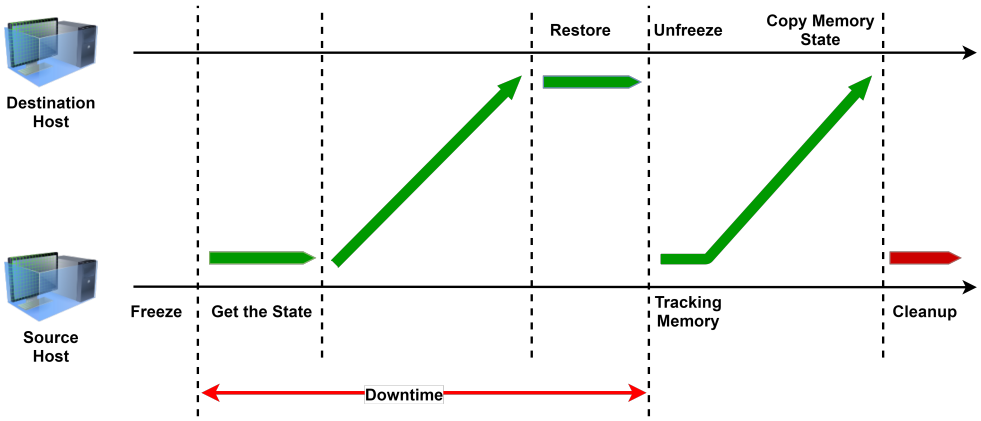


**Figure 4.** Phases of post-copy container live migration [9].

The post-copy and cold migration are not dependent on the rate of update of pages, but in pre-copy, this is then a factor affecting the performance. With respect to downtime, this will be more in cold and post-copy migration. Considering the quality of service, our model favors pre-copy because of its minimal downtime. Furthermore, different techniques are applied on pre-copy to reduce the re-transmission of pages, and can be further reduced.

#### 1.2.4. Hybrid Migration

As highlighted in the discussion in Sections 1.2.2 and 1.2.3, both the migration schemes have their own limitations. As an alternative, combining both techniques is known as hybrid migration. Initially, the source container's state is pre-dumped and transferred while the source is still operating. The source container is then stopped at the source, the current state is dumped and transmitted to the destination to restore the container. When the container is restored at the destination, it only needs the modified pages instead of all the pages.

Compared to the pre-copy approach, the rate of update of pages will not affect the downtime. It affects only the total migration time. The drawback is still the same as was discussed for post-copy, but the chances of such cases are rare in this approach. For the four types of migration techniques, it is imperative to understand what type of data is transferred. To understand this, first, it is necessary to know the phases of the migration techniques, as shown in Figure 5. In this context, we focus on three main phases: (pre-dump, dump, and demand-paging).

1. Cold migration: Nothing will be migrated in pre-dump. However, in the actual dump, the state of the container and memory pages are transferred and complete the migration process.
2. Pre-copy: The state of the container and the memory pages are transferred in pre-dump, and the iterative phase is initiated to send the dirty pages.
3. Post-copy: There is nothing to transfer in the pre-dump phase and the execution state is transferred in the dump phase. According to the request generated, all the memory pages will be transferred in the demand-paging phase.
4. Hybrid: The state of the container and the memory pages are transferred in pre-dump, and the iterative phase is initiated to send the dirty pages. It also transfers the dirty pages in demand-paging.

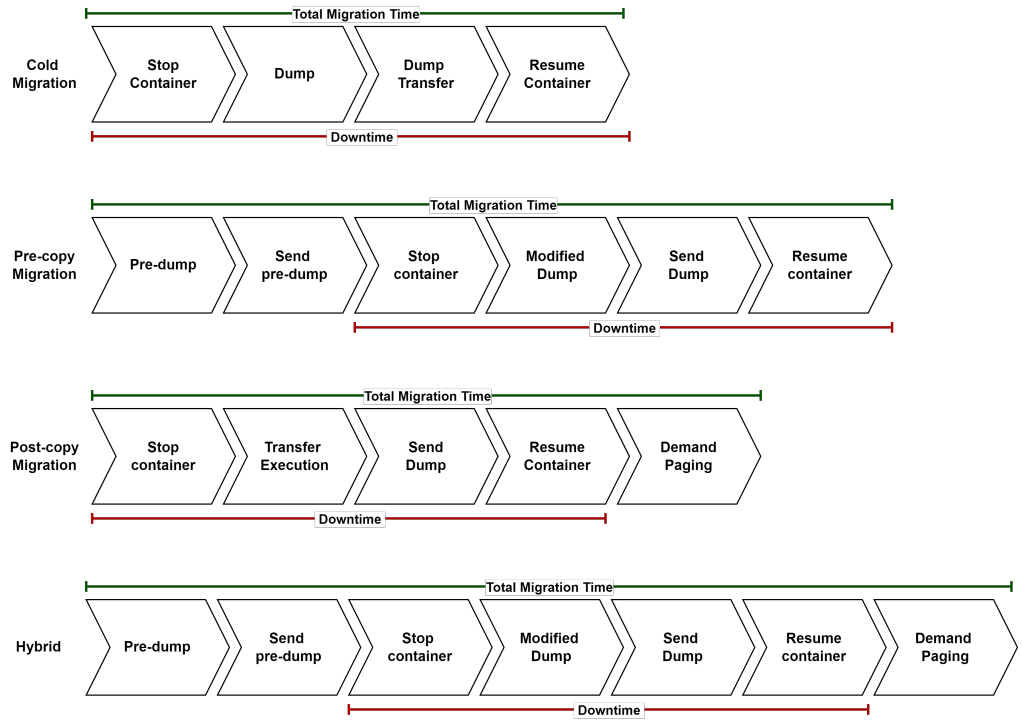


**Figure 5.** Phases of different migration techniques of container [10].

The actual number of phases is different depending on the type of migration. The total migration time and downtime depend on the phases.

The proposed pre-copy technique addresses the issue of container migration. The main objective is to minimize the amount of data transfer during the iterative dump.

### *1.3. Checkpoint and Restore*

The process of storing the runtime state of any running application and restoring this at a later stage is called checkpoint and restore. It can be performed on the same machine or at a remote host of IIoT. This technique is used to recover from unusual crashes during the runtime of sensitive applications in Industry 4.0. After a particular time gap, it automatically stores the application's state.

In the last few years, demand for containerized applications has been increasing continuously and is widely accepted in cloud computing [11,12]. Checkpoint and restore is now very useful in container migration to make secure migrations. This technique is introduced by checkpoint and restore in userspace (CRIU). Live container migration is successful only with CRIU.

According to the open container initiative (OCI), the migration process of the container has two main parts: container runtimes and container images, as shown in Figure 6. To manage these, two container runtime interfaces (CRI) are required. The CRIs support RunC and the container daemon. Daemon takes care of storage and libraries, while, RunC handles the container runtime. Daemon will generate the pull request for the required container images from a registry. They work together to obtain the required libraries and manage the container runtime. RunC can initiate the container process by communicating with the host kernel. It passes the command to the kernel to startup the process in a particular namespace, Pid, Cgroup, etc.

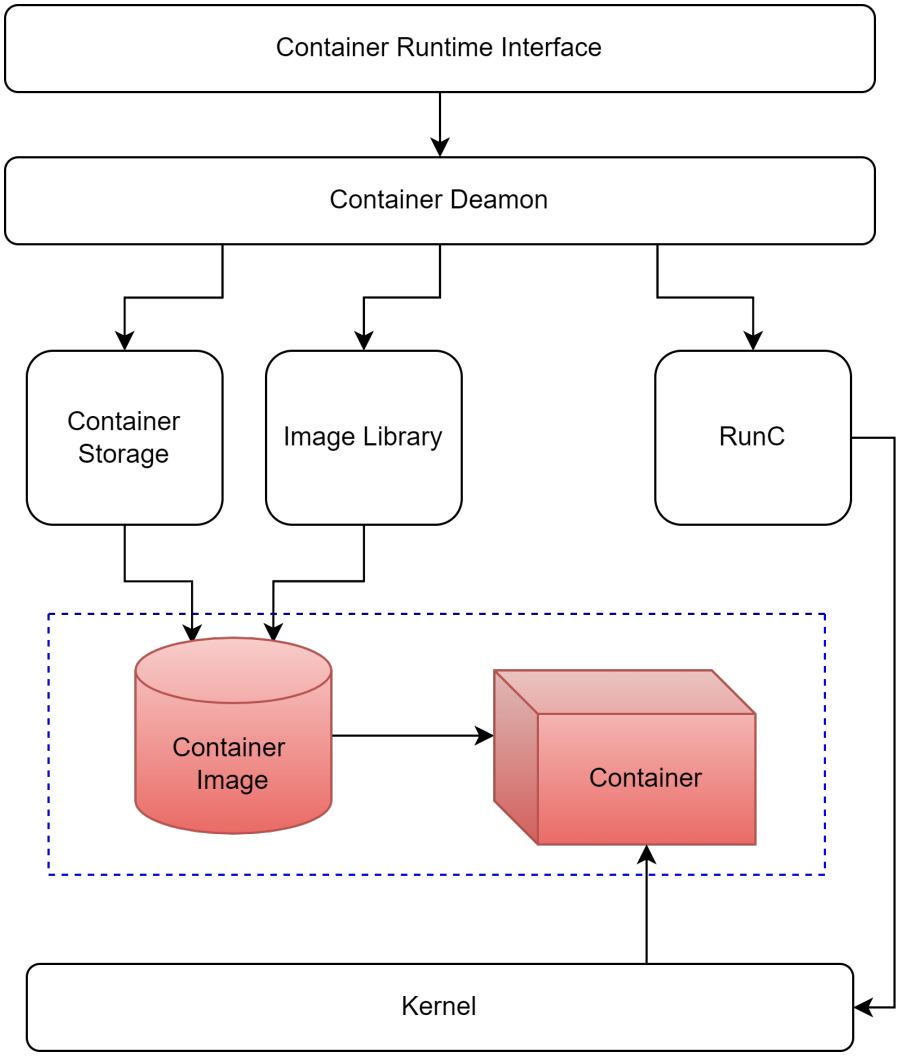


**Figure 6.** Container run-time interface.

### *1.4. Long Short-Term Memory*

In sequence prediction tasks, long short-term memory (LSTM) networks are utilised to learn order dependence. This has been shown to overcome the RNN's vanishing gradient constraints [13]. LSTM is ideally suited to forecasting for time series data because of its capability. The LSTM network is made up of a series of interconnected LSTM units/cells.

The cell state, shown as dotted lines in Figure 7, is the most significant component of the LSTM because the data from the gates is retained. The LSTM is divided into three layers known as gates: forget, input and output. The forget gate decides whether the data coming from a previous timestamp is relevant to remember or relevant to forget. The input gate tries to learn and provide new data to the cell. The third is the output gate, which passes the data for the current cycle to the next timestamp.

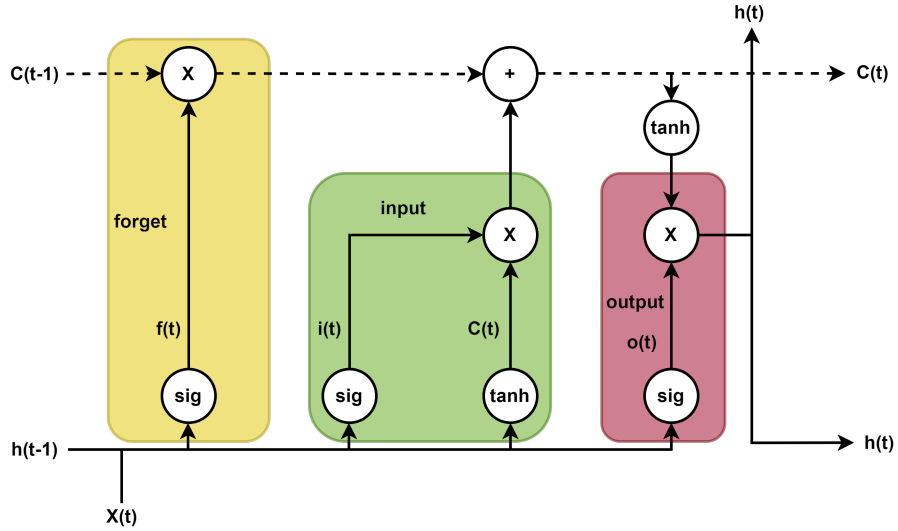


**Figure 7.** Architecture of long short-term memory [14].

LSTM has a hidden state as well, where $h(t-1)$ represents the hidden state of previous iteration and $h(t)$ provides the hidden state of the current iteration. Both these handle "short-term memory". In the same way, the cell state of the previous iteration is $C(t-1)$ and for the current iteration is $C(t)$. The cell states handles "long-term memory". The unit processes the input data for each input vector to the LSTM network as follows:

1. A new vector will be created by adding the $h_{t-1}$ (hidden state vector) and the $x_t$ (input vector). The newly created vector will be used as input to the *tanh* function and the three gates.
2. The flow of previously stored cell states is regulated by the forget gate:
   $f_t = sig(W_{hf} * h_{t-1} + x_t * W_{xf} + b_f)$
   where, $W_{hf}$ is the weight of the previous hidden state $h_{t-1}$ and $W_{xf}$ is the weight of the input. $x_t$ is the input at timestamp $t$ and $b_f$ is the bias parameter. Further, by applying a sigmoid function, the $f_t$ ranges in between 0 and 1. If $f_t = 0$, then forget everything and if $f_t = 1$ that means forget nothing.
3. The input gate quantifies the data from input:
   $i_t = sig(W_{hi} * h_{t-1} + x_t * W_{xi} * b_i)$
   where, $W_{hi}$ is the weight of input for the hidden state $h_{t-1}$ and $W_{xi}$ is the weight of the input. $x_t$ is the input at timestamp $t$. The sigmoid function manages the input value between 0 and 1.
4. The new data needs to be passed to the cell state:
   $N_t = tahn(W_{hc} * h_{t-1} + x_t * W_{xc} + b_c)$
   Here, the activation function is *tahn*, which manage the value of $N_t$ between $-1$ to 1. If it is positive, then data will be added to the cell state and if negative, then the data is discarded from $C_t$.

5. The final calculated $C_t$ is:
   $C_t = f_t * C_{t-1} + i_t * N_t$
6. The output gate determines how much $C_t$ is passed to the next cell. The hidden state $h_t$ is calculated as follows:
   $o_t = sig(W_o * h_{t-1} + x_t * b_o))$
   $h_t = o_t * tanh(C_t)$

The output gate is further subdivided. The sigmoid function is applied to the filter $h(t-1)$, $h(t)$ and these are used to scale the values of the vector generated from the tanh cell, which manages the values from $-1$ to 1. Then, the product of the filter mentioned above and the vector is the output state for the next cell state.

The main focus of this paper is on container migration. We surveyed migration techniques and point out their advantages and disadvantages [15].

1. The pre-copy migration technique is chosen to perform live migration. It is preferred for live container migration, but the factor affecting its performance is the amount of data transfer in the iterative dump. The proposed work is carried out on the iterative phase only. The pre-dump phase is discussed in our previous research paper.
2. A predictive container migration technique is proposed for the iterative dump to minimize the data transfer over the network.
3. A prediction model is designed and implemented with LSTM.
4. Consideration of experimental results implemented in different scenarios and compared with other migration techniques shows that the proposed system outperforms other techniques.

The rest of the paper is organised as follows: Section 2 describes the literature review and the research gap. The problem identification along with the main objective of the study is discussed in Section 3. The tools and techniques used in the proposed system and the prediction model using LSTM are discussed in Section 4. In Section 5, the evaluation of the system model is elaborated in detail and concluded in Section 6.

## 2. Related Work

The live migration of containers is prevalent in the IIoT ecosystem, and most Industry 4.0 cloud services are switched to this platform. With the increasing popularity and wide adoption of containers, challenges also arise. The container's size is already smaller compared to virtual machines, but the performance of containers can be enhanced further. The most influential factor is the data migration over the network, which directly affects the cost and performance of container migrations in the lightweight IIoT environment. There are several techniques proposed to minimize the dump size.

C Puliafito et al. [8] have carried out a detailed evaluation of various migration techniques under four parameters including total migration time, downtime, dump time, and amount of data transferred. They recognized several types of situations and suggested which strategy would be most suited to them. The findings demonstrated that cold migration suffers from significant downtime, whereas hybrid migration suffers from a longer overall migration time. Pre-copy and post-copy migrations may thus be the best alternatives under specific scenarios.

Several common memory compression methods have been reported and studied from a memory standpoint, including RLE, Huffman Coding, and a novel strategy for minimizing migration time. There are numerous difficulties regarding memory compression, and an effective scheduling strategy is suggested [5], but the compression overhead affects migration time. To provide load balancing, the container is scheduled on the relevant server. The core implementation of dynamic migration scenarios has been developed for IoT [16], with the main motive of resource provisioning. It can be more effective if it is managed to reduce memory transfer. A successful migration is needed to restore huge data structures following a disaster. Moreover, to reduce the data transfer, prediction methods can be applied [17]. It can be reduced further with LSTM as suggested by [18]. The reuse distance concept is used, and the changed memory pages are traced back during

the copy process [19]. The model includes clusters, containers, and micro-resources with four optimization goals. The experimental results showed that this strategy provided a solution to the issue of container allocation and flexibility, attaining higher ethical standards than the Kubernetes container management regulations. Each method's implementation scenarios are examined in [20]. For containers, pre-copying is the predominant means of migration.

Elghamrawy et al. [21] discussed the fact that there is a significant discrepancy between distinct prediction systems in the behavior of current memory pages. That is the hole they wanted to fill. They characterized the behavior of memory pages using a prediction approach for relatively stable memory pages and using the memory page characterization to prioritize specific pages with live migration since these pages will be updated gradually in subsequent cycles. The genetic algorithm technique employing the non-dominated sorting genetic algorithm is recommended to maximize container assignment and management elasticity to the degree that this algorithm has produced good results for other cloud management challenges.

Mirkin et al. [22] developed the OpenVZ container checkpoint provided with a resume mechanism. This function allows the container to scan and restart programs and network connections. Checkpoints and restarts work directly with the kernel to reduce service delays and the size of the dump file.

Dump size can be further reduced for memory intensive applications. Molto et al. [23] designed a hybrid distributed computing method for VM's and containers for better synchronization among hosts. In memory intensive applications, the repetition of memory may increase.

The live container just-in-time migration service was designed in accordance with OCI (Open Container Initiative) standards introduced by Nadgowda et al. [24]. Containers run on the server with a shared file system and some with the local file system. CRIU helps to perform live migration of the container and ensures restart on the destination host. A union mount file system and CRIU help to reduce the migration time and data transfer.

Luo et al. [25] developed a technique based on data compression and de-duplication. The authors employed the RLE technique and the runtime storage image identity to reduce duplicate memory data. Hash-based fingerprints were employed for page similarity calculations. The LRU hash tables FNHash and FPHash were used for implementation. The efficiency of migration increased in terms of space with the overhead of CPU resources.

During the incremental copying procedure, the primary approach of pre-copy transfer memory pages, which are duplicated repeatedly at high replacement rates, is provided by Ansar et al. [26]. The article described an optimized pre-copy method (OPCA) with a Gray–Markov model. They shortlisted the pages on the basis of modification rate, which decreased the number of iterations and other parameters. This increased the resource utilization which was managed by using a hot and warm working set to categorize the pages. Now, only the pages from the HW set will go through the process. Chronopoulos et al. [27] showed effective use of machine learning to build an artificial neural network for speech language therapy.

There are other techniques as well which are suitable for the prediction of memory. Sobia Pervaiz et al. [28] conducted a detailed review of variants of PSO and highlighted the key features of each variant. This contributed to identifying the best suited variant of PSO depending on the nature of the problem. Waqas Haider Bangyal et al. [29] used a unique quasi-random sequence, termed the WELL sequence, to initiate the PSO particles. The velocity and position vectors of the particles were changed in a random order. According to the results obtained, the WE-PSO strategy outperformed the PSO, S-PSO, and H-PSO approaches. The authors of [30] presented three sequence strategies: Torus, Knuth and WELL. All the techniques were tested with low-discrepancy sequences. The results showed that the proposed techniques outperformed the standard PSO and its other variants.

Moving a container application can be accomplished using various container migration techniques, and, to increase the performance, various alternatives are available, as shown in Table 1. Because of the container's limited lifespan, a pre-copy strategy to facilitate the migration procedure was used [31].

**Table 1.** A prediction-based comparison of various container migration techniques.

| Ref. | Migration Technique | Prediction Method | Achieved | Outcome |
|---|---|---|---|---|
| [5] | pre-copy | RLE, Huffman Coding | memory compression | compression overhead effects migration time |
| [16] | CloudIoT | LXC | vertical offloading | main motive is resource provisioning |
| [17] | pre-copy | ARIMA | predict dirty pages, reduce and compress | can be reduced further with LSTM [18] |
| [19] | pre-copy | ARIMA | memory forecast | can be reduced further using containers with LSTM [18] |
| [20] | pre-copy | LXD | container resource management | resource provisioning |
| [22] | pre-copy | OpenVZ | incremental checkpoint | dirty pages transmission can be minimized |
| [31] | pre-copy | Gray-Markov prediction model | reduce iterative cycle | it shortlist the active pages |
| [18] | pre-copy | LSTM and ARIMA | dirty page prediction | 600 times faster prediction time than ARIMA |
| Proposed Approach | pre-copy | LSTM | reduce the size of iterative dump | amount of data transfer is reduced by 31.04% |

## 3. Motivation and Research Gap

Containers provide a robust environment for virtualized computing. The choice of container migration mechanisms significantly impacts container migration performance. The various strategies were covered in [15]. The comparison demonstrated that pre-copy and hybrid-copy migration approaches outperformed post-copy migration techniques. This work was separated into three steps based on the pre-copy live migration technique: pre-dump, iterative dump, and final dump. The container was running during these phases in order to accomplish live migration for Industry 4.0 applications.

When it is decided to migrate a container using the existing technique of pre-copy container migration, then all the associated memory pages and their configuration are sent to the destination host in the pre-dump phase. Further, in the iterative phase, all the updated pages will be migrated in every iteration, and there is a chance that a single page may be migrated several times. This overhead directly impacts several parameters, including the performance and the operating cost. The data transfer in an iterative phase is hugely dependent on the size of the container and the kind of operation it is handling. Sometimes containers send data which are greater than the actual data size due to the re-transmission of updated pages. This is the main factor affecting the performance. Our main objective was to lower the quantity of data transfer across the network during the iterative phase.

We designed an algorithm for the iterative phase of pre-copy container migration to minimize page transfer for cross-domain and cross-border IIoT applications. In this algorithm, instead of sending updated pages in every iteration, we predicted the chances of modification in the subsequent few iterations to minimize re-transmission. If sending the active pages in the last few iterations is possible, then the data transmission can be minimized.

CRI handles all the container activities, such as start, manage and stop. If we focus on container migration for any reason, then the main concern is about container image and runtime, as highlighted in Figure 6. When there is a requirement to start a container, CRI takes control, and the CRI daemon and Runs work together to initiate a container. There are a total of three phases in pre-copy container migration.

To understand the scope for improvement, let us discuss the iterative phase at the memory level. When a container runs on a machine, some memory is allotted to that container. These memory pages are in use by the container and are modified frequently. When it is decided to migrate a container to a target host, the whole container memory set is transferred to the destination host at the initial state of migration. The same set of

memory pages will be migrated repeatedly to synchronize the destination host memory with current changes on the source host, as illustrated in Figure 8.

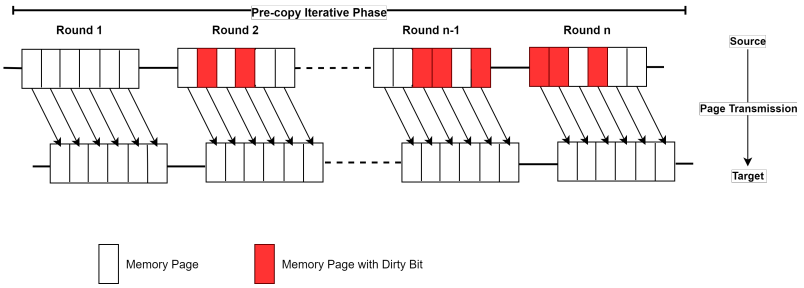


**Figure 8.** Process of transferring pages in an iterative phase of existing approach.

Nowadays, Industry 4.0 widely adopts containers, and researchers are also working on this. If we consider the latest version of pre-copy migration, then there is no need to re-transfer the same memory page in the iterative phase. Depending on the data updated in the last iteration, it can be decided whether a page will be migrated to its destination or not, as shown in Figure 9. So, in this Figure, the iterative phase is explained with the number of rounds. Each round handles a set of memory pages represented with a rectangular box and subdivided into individual pages. Further, the pages have categorizations including:

1. Pages with no color denote no updating compared to the previous iteration.
2. Pages with red color denote that they are updated after the previous iteration.

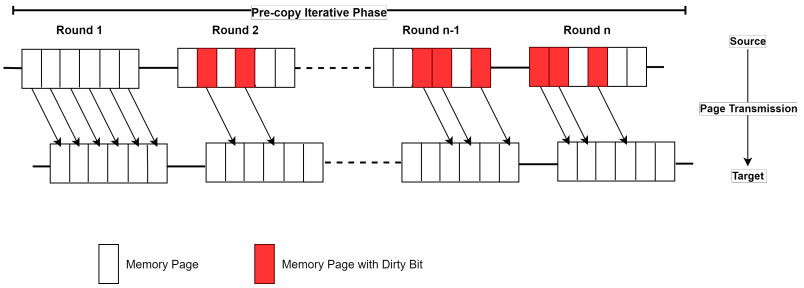


**Figure 9.** Process of transferring pages in an iterative phase according to dirty bits in the existing approach.

In Figure 9, it can be seen that, in round one, all the pages are transferred from source one to the target host because there was no updating in the memory pages. If check round 2 is checked, two pages have dirty bits from this chunk of pages. Only these updated pages will be migrated to the destination host, whereas other pages will be discarded from this re-transfer. The exact process will be followed in all iterative rounds, and this helps to minimize the data transfer during this iterative phase of pre-copy container migration.

We have conducted a detailed review of the container migration techniques and identified the scope for enhancement in the iterative phase. We can further reduce the data transmission during the iterative phase. This method is based on the concept of the dirty bit and the rate of updating of pages. As can be seen in Figure 10, in rounds 1 to 3, there are no pages transferred from source to target host. It will wait for the first three iterations to analyze which page will be migrated to the source host and which page will be discarded.

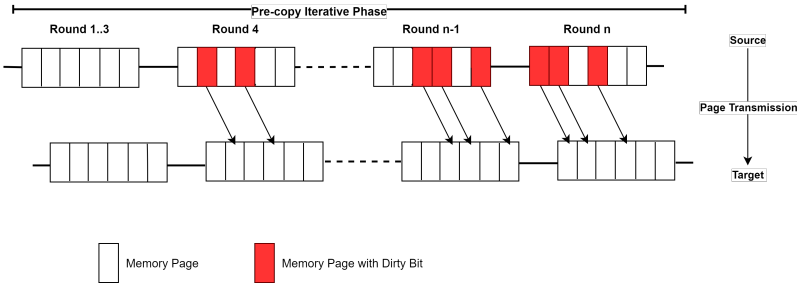


**Figure 10.** Process of transferring pages in an iterative phase according to dirty bits prediction in the proposed approach.

The pages that need to migrate are shortlisted using a record of the last three iterations. If a particular page is being updated, then discard that page from the dump, and if a page is not updated in the last three iterations, it will migrate to the destination host. This has a significant impact on memory-intensive application containers. It will minimize the cost over the network by minimizing the data transfer to the destination host.

## 4. Methodology

A new algorithm was designed to help to minimize the dump size. In this Algorithm 1, the first three iterations will only record the activities of every page. Starting from the fourth iteration, a decision will be taken according to the activity status of the last three iterations. If a page is not modified in the last three iterations, that page will be added to $S_{pool}$, otherwise the page remains in the active set. This process will be repeated for all the remaining iterations. Then the status of the pages will be provided to the proposed prediction scheme for a final decision of the iterative dump.

**Algorithm 1:** Iterative Dump/Checkpoint.

**Result:** Prediction of memory pages to be migrated in the iterative phase
Access the memory pool of the container ;
Get the read-write status of every memory page in every iteration;
**while** $M_{pool}$ **do**
  **if** $DB(P_i) == 1$ **then**
    $P_i.active+ = 1$;
    ▷ Page will be added to active pool
  **else**
    **if** $P_i.active \geq 1$ **then**
      $P_i.active = 0$;
    **else**
      $P_i.active- = 1$;
    **end**
  **end**
  **if** $P_i.active \leq -2$ **then**
    $S_{Pool}.append(Pi)$;
    ▷ Page will be added to send pool
  **end**
**end**

*Machine-Learning-Based Predictive Checkpoint*

The LSTM is used in a network model to predict the memory load to be migrated. In this network model, the input cells layer are used as 3, the LSTM layer units are 10, and 1 is the output layer cell as shown in Figure 11. The input layer of this network model receives the modification data of the previous three iterations to predict the chances of modification

in the next iteration. This model aids in generalizing the training data, allowing adjustment of the input size.

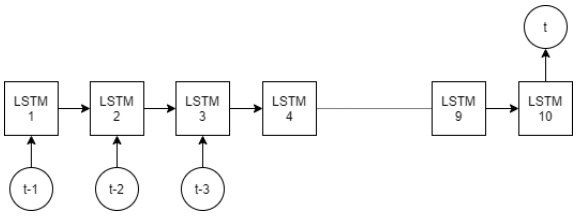


**Figure 11.** LSTM Network Architecture [32].

The architecture of the prediction model was designed using LSTM which works on the page history generated by the prediction model. This model works with three inputs which interact with the hidden layer with a size of 10 units and finalizes the result with a single output layer; the parameter descriptions are provided in Table 2. This works in a serial order after the process shown in Figure 12. The loss function for training the LSTM prediction models is the mean square error. The sample input is divided into batch sizes of 64 which works with a maximum of 10 iterations. Then these iterations with the weighted input will be further processed in each epoch. To set the initial random weight, the initializer used is "RandomUniform" and the bias initializer is "zeros".

**Table 2.** LSTM Model Configuration Parameters

| Parameters | Range |
|---|---|
| LSTM units | 10 |
| Kernel initializer | Random uniform |
| Loss function | MSE |
| Batch size | 64 |
| Size of input | 3 |
| Layers of LSTM | 1 |
| Size of output | 1 |
| epochs | 10 |

The suggested approach significantly minimizes the size of the checkpoint. In this approach, the active set of pages minimizes the re-transmission of pages because the incremental checkpoint ignores frequent copying of dirty pages and transmission. It has a direct impact on application downtime. As a result, delayed checkpoint approach is proposed based on the prediction of dirty pages, which can reduce application downtime by reducing the time spent regularly copying and transmitting dirty pages. A detailed explanation of the proposed prediction model, as shown in Figure 12, is provided below:

1. In every iteration of the prediction model, the activities related to the memory pages are recorded and stored in "Page modification history".
2. On the basis of memory status, Algorithm 1 will shortlist the pages in $send_{pool}$.
3. These pages will be further passed to the input gate of the LSTM module which works as shown in Figure 7. It works on the provided input according to the mentioned epochs and generates an output with a timestamp.
4. Depending on the condition of the iterative-dump, if the stopping criteria does not match, then the next iteration is initiated.
5. Now, the iterative algorithm again fetches the latest status of the memory pages and passes it to LSTM module. The previous timestamp output is also provided to the current state of the LSTM.
6. In every iteration, the sorted pages after the LSTM module are migrated to the destination host.
7. When stopping criteria matches, then the control shifts to the final-dump of the pre-copy migration technique.

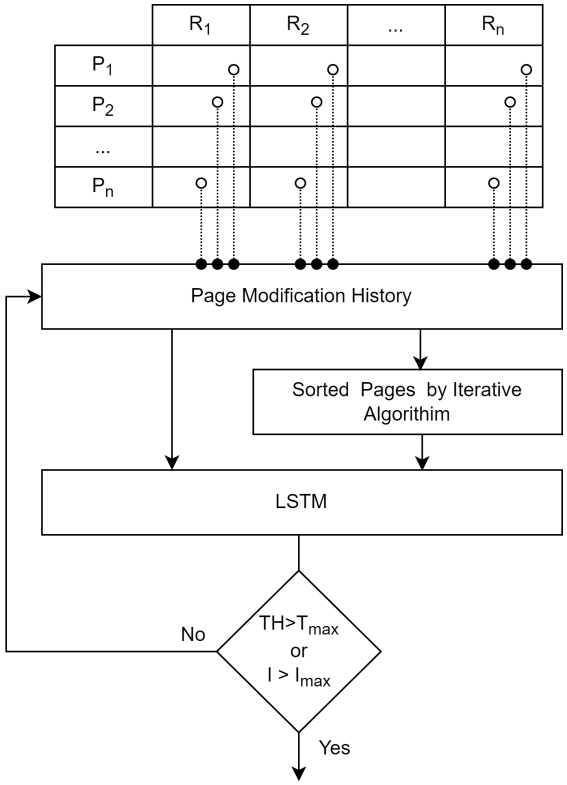


**Figure 12.** The proposed prediction model to shortlist the pages to be transferred.

The parameters $(P_1, P_2 \ldots P_n)$ represent pages related to a container, and $(R_1, R_2 \ldots R_n)$ represents the iteration number. During this iterative phase, the dirty bit status of each page in every iteration is stored in "Page History". The same input will be provided to an iterative algorithm to sort the pages. Then both the inputs will be provided to the proposed LSTM module to predict the final set of pages to be migrated in the next iteration. The total number of iterations depends on the dynamic conditions, such as if the updating rate of memory pages reaches the set threshold level or when the iteration count reaches the specified maximum limit, then the iterative phase ends immediately. The maximum threshold value "$T_{max}$" is set as 70% and the maximum iterations "$I_{max}$" are 10.

## 5. Results and Discussion

This prediction model was developed using a direct multi-step process to forecast the number of dynamic pages to be migrated in the following round. Cloudsim 4.0 libraries are used to simulate the environment for container migration and the Keras library manages the implementation of these prediction models with the help of DL4J. LSTM is used to predict short-term memory changes. The tensor processing unit (TPU) is used to train the model. The mean squared error (MSE) is used to measure the degree of errors. If there is no error, the MSE generates a value of 0. By calculating the average time to predict after ten attempts, the model will be examined using the mean square error. There are some container migration techniques. Figure 13 provides a comparison of various container migration techniques with a batch of five containers where: (a) shows the total amount of data transfer during the iterative dump, (b) shows the time to transfer the iterative dump, (c) indicates the overall downtime, including resume time on the destination host, and (d) shows the total time taken during the migration process.

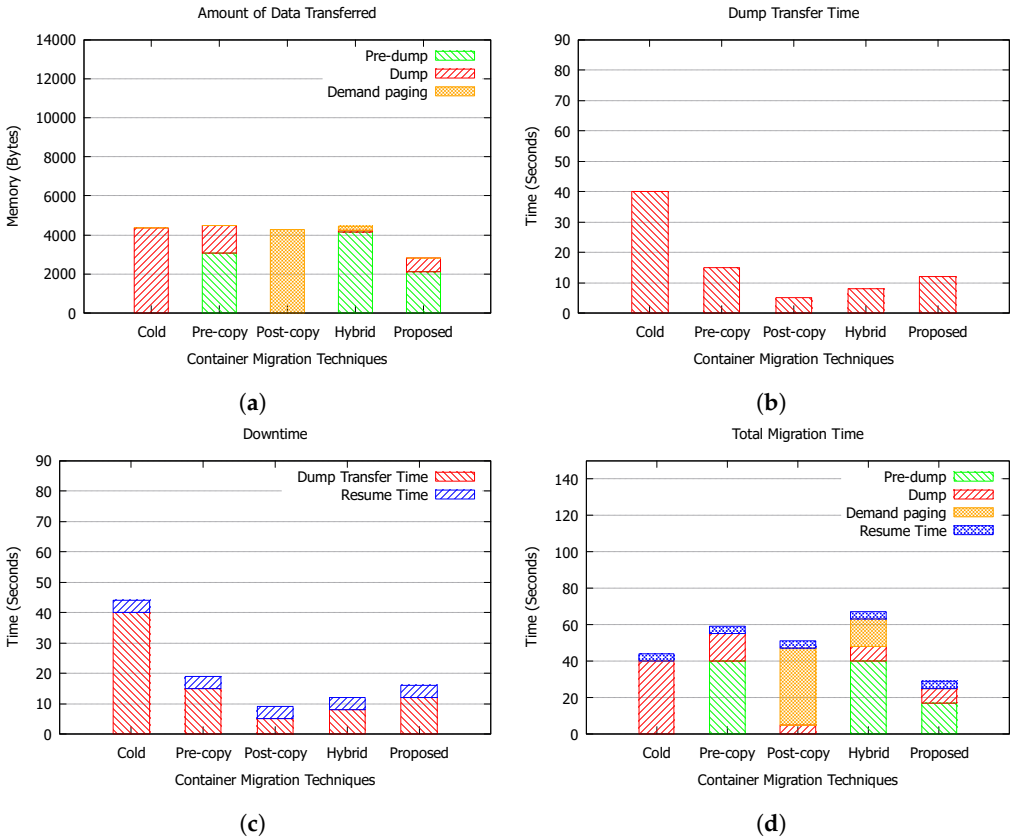


**Figure 13.** Comparison of various container migration techniques with batch of five containers where (**a**) shows the total amount of data transfer during iterative dump, (**b**) shows the time to transfer the iterative dump, (**c**) indicates the overall downtime of a container including resume time on destination host and (**d**) shows the total time taken during the migration process.

We implement the proposed container migration technique as a pre-copy and compare it to the existing pre-copy; the amount of data transfer is reduced in the proposed approach, while the dump time downtime, and the total migration time are also minimized.

Figure 14 shows a comparison of various container migration techniques with a batch of five containers, where: (a) shows the total amount of data transfer during the iterative dump, (b) shows the time to transfer the iterative dump, (c) indicates the overall downtime, including the resume time on the destination host, and (d) shows the total time taken during the migration process. We implement the proposed container migration technique as a pre-copy. Compared to the existing pre-copy, the amount of data transfer is reduced in the proposed approach, and the dump time downtime, and total migration time are also minimized.

The results shown in Figure 15 indicate almost the same ratio as discussed in the previous two scenarios with batch size 5 and 10 containers. The pooled results of all three batches show that the proposed technique outperformed the existing pre-copy. The post-copy and hybrid technique had downtime and dump time less than the proposed technique. In these techniques, some of the processes are carried out after migration, which increases the amount of data transfer and the total migration time, as shown in Figure 15a,d.

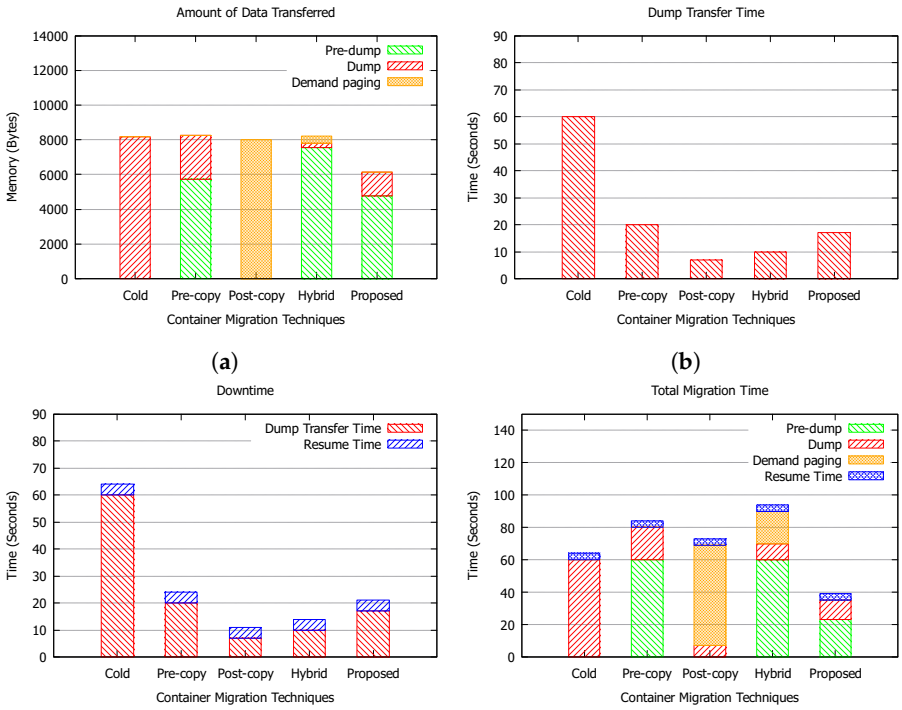


**Figure 14.** Comparison of various container migration techniques with batch of 10 containers, where (**a**) shows the total amount of data transfer during iterative dump, (**b**) shows the time to transfer the iterative dump, (**c**) indicates the overall downtime of a container including resume time on destination host and (**d**) shows the total time taken during the migration process.

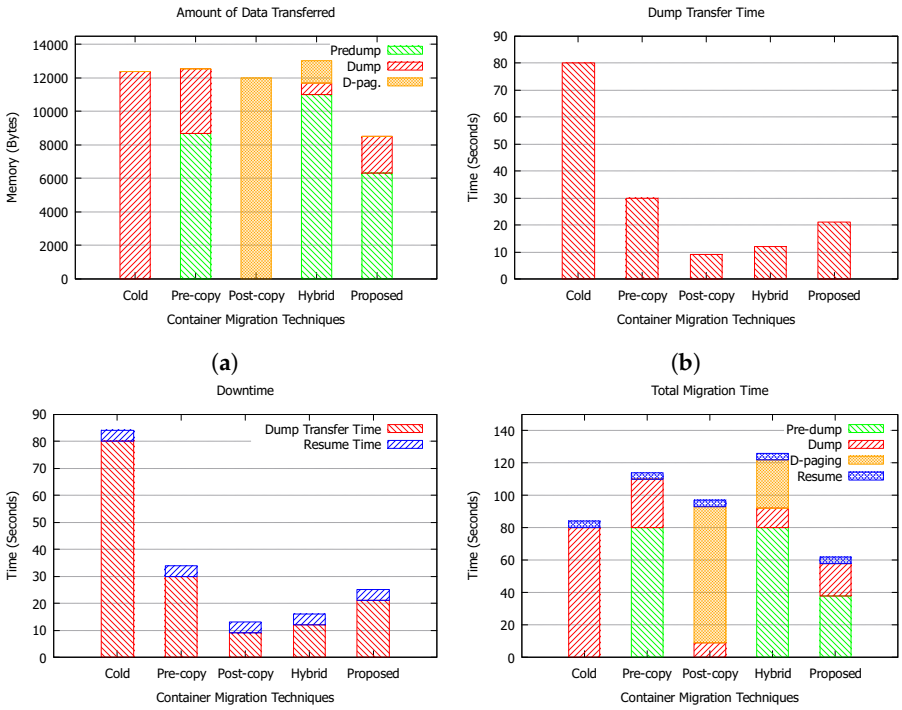


**Figure 15.** Comparison of various container migration techniques with batch of 15 containers where (**a**) shows the total amount of data transfer during iterative dump, (**b**) shows the time to transfer the iterative dump, (**c**) indicates the overall downtime of a container including resume time on destination host and (**d**) shows the total time taken during the migration process.

## 6. Conclusions

Application deployment on virtual machines is more efficient than on native servers, and switching to containers can provide even more value in the IIoT ecosystem. The increasing demand for containers makes them very popular in IT organizations. To increase the performance of Industry 4.0, we have implemented container migration in different batches of containers. These batches are categorized as a set of 5, 10, and 15 containers. In this paper, our approach minimizes the data transfer over the network. We have implemented the proposed approach as pre-copy migration. The results show that the proposed approach outperforms existing approaches regarding the amount of data transferred and total migration time. With respect to the time taken to transfer the dump, the proposed technique takes less time than cold and pre-copy migration. However, the time taken by post-copy and a hybrid approach is less than the proposed approach. Similarly, in the case of downtime, the cold and pre-copy approach downtime is more than the proposed approach, but the post-copy and hybrid have less downtime, because in both post-copy and hybrid, most of the data is transferred after the container is initiated on the destination host. The proposed technique outperforms for "total migration time" and "amount of data migrated", but post-copy gives better results for the remaining two parameters (downtime and dump-time) as shown in Table 3.

**Table 3.** Comparison of various container migration techniques on the basis of migration time, downtime, dump time and amount of data transfer.

| | Total Migration Time | Downtime | Dump Time | Data Migrated |
|---|---|---|---|---|
| Cold | 64.00 | 64.00 | 181.00 | 8310.00 |
| Pre-copy | 85.67 | 25.67 | 65.00 | 8427.67 |
| post-copy | 73.67 | 11.00 | 21.00 | 8091.00 |
| Hybrid | 95.67 | 14.00 | 28.00 | 8556.67 |
| "Proposed Pre-copy" | 43.33 | 20.67 | 51.00 | 5811.33 |

Nevertheless, compared to pre-copy only:

1. Total migration time is decreased by 49.42%
2. The downtime is decreased by 19.47%.
3. The dump-time is minimized by 21.53%.
4. The amount of data transfer is reduced with 31.04%

In future research, the memory prediction in live container migration can be evaluated at large scale in real scenarios to obtain detailed insights. Because of increasing demand for containers, the amount of data transfer is a crucial factor for the overall performance. The type of application running in the container may effect the prediction results. We plan to extend our study on different type of applications and to test these with alternative prediction schemes, such as ANN, ARIMA etc. to determine future directions.

**Author Contributions:** Conceptualization, G.S. and P.S.; methodology, G.S.; software, S.S.A.; validation, P.S., M.H. and M.M.; formal analysis, G.S.; investigation, G.S.; resources, M.H.; data curation, M.M. and S.S.A.; writing—original draft preparation, G.S.; writing—review and editing, P.S. and M.M.; visualization, G.S. and M.H.; supervision, P.S. and M.M.; project administration, M.H.; funding acquisition, M.M. and S.S.A. All authors have read and agreed to the published version of the manuscript.

**Funding:** Taif University Researchers Supporting Project number (TURSP-2020/215), Taif University, Taif, Saudi Arabia.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Acknowledgments:** The authors are grateful for the support of Taif University Researchers Supporting Project number (TURSP-2020/215), Taif University, Taif, Saudi Arabia.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
2. Zhang, Q.; Liu, L.; Pu, C.; Dou, Q.; Wu, L.; Zhou, W. A comparative study of containers and virtual machines in big data environment. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 178–185.
3. Mellado, J.; Núñez, F. Design of an IoT-PLC: A containerized programmable logical controller for the industry 4.0. *J. Ind. Inf. Integr.* **2022**, *25*, 100250. [CrossRef]
4. Nadgowda, S.; Suneja, S.; Bila, N.; Isci, C. Voyager: Complete container state migration. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2137–2142.
5. Al-Dhuraibi, Y.; Paraiso, F.; Djarallah, N.; Merle, P. Autonomic vertical elasticity of docker containers with elasticdocker. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, 25–30 June 2017; pp. 472–479.
6. Ma, L.; Yi, S.; Li, Q. Efficient service handoff across edge servers via docker container migration. In Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose, CA, USA, 12–14 October 2017; pp. 1–13.
7. Prakash, C.; Mishra, D.; Kulkarni, P.; Bellur, U. Portkey: Hypervisor-assisted container migration in nested cloud environments. In Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, online, 1 March 2022; pp. 3–17.
8. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F. Design and evaluation of a fog platform supporting device mobility through container migration. *Pervasive Mob. Comput.* **2021**, *74*, 101415. [CrossRef]
9. Containers Live Migration Behind the Scenes. Available online: https://www.infoq.com/articles/container-live-migration/ (accessed on 25 September 2021).
10. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F.; Puliafito, A. Container migration in the fog: A performance evaluation. *Sensors* **2019**, *19*, 1488. [CrossRef] [PubMed]
11. Abdulsalam, Y.S.; Hedabou, M. Decentralized Data Integrity Scheme for Preserving Privacy in Cloud Computing. In Proceedings of the 2021 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC), Chengdu, China, 18–20 June 2021; pp. 607–612.
12. Bentajer, A.; Hedabou, M. Cryptographic key management issues in cloud computing. *Adv. Eng. Res.* **2020**, *34*, 78–112.
13. Understanding LSTM Networks. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 22 November 2021).
14. Tang, J.; Li, Y.; Ding, M.; Liu, H.; Yang, D.; Wu, X. An Ionospheric TEC Forecasting Model Based on a CNN-LSTM-Attention Mechanism Neural Network. *Remote Sensing* **2022**, *2022*, 14102433. [CrossRef]
15. Singh, G.; Singh, P. A Taxonomy and Survey on Container Migration Techniques in Cloud Computing. In *Sustainable Development Through Engineering Innovations*; Springer: Singapore, 2021; pp. 419–429.
16. Dupont, C.; Giaffreda, R.; Capra, L. Edge computing in IoT context: Horizontal and vertical Linux container migration. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–4.
17. Patel, M.; Chaudhary, S.; Garg, S. Improved pre-copy algorithm using statistical prediction and compression model for efficient live memory migration. *Int. J. High Perform. Comput. Netw.* **2018**, *11*, 55–65. [CrossRef]
18. Imdoukh, M.; Ahmad, I.; Alfailakawi, M.G. Machine learning-based auto-scaling for containerized applications. *Neural Comput. Appl.* **2020**, *32*, 9745–9760. [CrossRef]
19. Masdari, M.; Khezri, H. Efficient VM migrations using forecasting techniques in cloud computing: A comprehensive review. *Clust. Comput.* **2020**, *23*, 2629–2658. [CrossRef]
20. Bhardwaj, A.; Krishna, C.R. A Container-Based Technique to Improve Virtual Machine Migration in Cloud Computing. *IETE J. Res.* **2019**, *68*, 401–416. [CrossRef]
21. Elghamrawy, K.; Franklin, D.; Chong, F.T. Predicting memory page stability and its application to memory deduplication and live migration. In Proceedings of the 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Santa Rosa, CA, USA, 24–25 April 2017; pp. 125–126.
22. Mirkin, A.; Kuznetsov, A.; Kolyshkin, K. Containers checkpointing and live migration. In Proceedings of the Linux Symposium, Ottawa, ON, Canada, 23–26 July 2008; Volume 2, pp. 85–90.
23. Moltó, G.; Caballer, M.; Pérez, A.; de Alfonso, C.; Blanquer, I. Coherent application delivery on hybrid distributed computing infrastructures of virtual machines and docker containers. In Proceedings of the 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), St. Petersburg, Russia, 6–8 March 2017; pp. 486–490.
24. Patel, M.; Chaudhary, S.; Garg, S. vMeasure: Performance Modeling for Live VM Migration Measuring. In *Advances in Data and Information Sciences*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 185–195.
25. Luo, S.; Zhang, G.; Wu, C.; Khan, S.; Li, K. Boafft: Distributed deduplication for big data storage in the cloud. *IEEE Trans. Cloud Comput.* **2015**, *8*, 1199–1211. [CrossRef]

26. Ansar, M.; Ashraf, M.W.; Fatima, M. Data Migration in Cloud: A Systematic Review. *Am. Sci. Res. J. Eng. Technol. Sci. (ASRJETS)* **2018**, *48*, 73–89.
27. Chronopoulos, S.K.; Kosma, E.I.; Peppas, K.P.; Tafiadis, D.; Drosos, K.; Ziavra, N.; Toki, E.I. Exploring the Speech Language Therapy through Information Communication Technologies, Machine Learning and Neural Networks. In Proceedings of the 2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 21–23 October 2021; pp. 193–198. doi: 10.1109/ISMSIT52890.2021.9604553. [CrossRef]
28. Pervaiz, S.; Ul-Qayyum, Z.; Bangyal, W.H.; Gao, L.; Ahmad, J. A systematic literature review on particle swarm optimization techniques for medical diseases detection. *Comput. Math. Methods Med.* **2021**, *2021*, 5990999. [CrossRef] [PubMed]
29. Bangyal, W.H.; Hameed, A.; Alosaimi, W.; Alyami, H. A new initialization approach in particle swarm optimization for global optimization problems. *Comput. Intell. Neurosci.* **2021**, *2021*, 6628889. [CrossRef] [PubMed]
30. Bangyal, W.H.; Nisar, K.; Ibrahim, A.; Bin, A.A.; Haque, M.R.; Rodrigues, J.J.; Rawat, D.B. Comparative analysis of low discrepancy sequence-based initialization approaches using population-based algorithms for solving the global optimization problems. *Appl. Sci.* **2021**, *11*, 7591. [CrossRef]
31. Nie, H.; Li, P.; Xu, H.; Dong, L.; Song, J.; Wang, R. Research on optimized pre-copy algorithm of live container migration in cloud environment. In Proceedings of the International Symposium on Parallel Architecture, Algorithm and Programming, Haikou, China, 17–18 June 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 554–565.
32. Phyo, P.P.; Jeenanunta, C. Advanced ML-Based Ensemble and Deep Learning Models for Short-Term Load Forecasting: Comparative Analysis Using Feature Engineering. *Appl. Sci.* **2022**, *2022*, 12104882. [CrossRef]