Copyright Notice

Institute of Communication Networks and Computer Engineering
University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany
Phone: ++49-711-685-68026, Fax: ++49-711-685-67983
Email: mail@ikr.uni-stuttgart.de, http://www.ikr.uni-stuttgart.de

2019 International Conference on Networked Systems (NetSys)

Investigation of Uninterrupted Service Live Migration Using Software-Defined Networking

Keerthana Govindaraj*, Mamia Saha*‡, Alexander Artemenko*, and Andreas Kirstaedter‡
* *Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany*
Keerthana.Govindaraj, Mamia.Saha, Alexander.Artemenko}@de.bosch.com
Andreas.Kirstaedter@ikr.uni-stuttgart.de

***Abstract*—The vision of Industry 4.0 is to enable a highly dynamic and flexible manufacturing system. Nevertheless, an uninterrupted and reliable service needs to be ensured to fulfill the safety requirements of the industrial applications. The latest advancements in technologies such as wireless connectivity, hardware virtualization, application offloading, etc. aims to cater to most of these requirements. Moreover, a new paradigm called *Edge Computing* is becoming a preferred solution to fulfil the latency and availability requirements of the Industry 4.0 applications. However, the dynamic resource management and automated service provisioning remains to be an open challenge in an environment with constantly varying requirements.**

Service migration is a part of dynamic resource management that enables repositioning of a service from one computation entity, which is e.g., overloaded and cannot satisfy the user requirements, to another computation entity. An uninterrupted service live migration is necessary to satisfy the low latency and high availability requirements of mobile devices on the factory floor. In this paper we introduce and investigate a new approach for an uninterrupted service live migration.

***Index Terms*—zero downtime, live migration, SDN, Edge Computing**

I. Introduction

Smart factories [1] or Industry 4.0 reflects a cost-effective manufacturing set-up with fast and flexible reconfiguration of production processes. This new level of automation requires applications such as augmented reality, collaborative robotics, automated transportation, etc. However, these applications depend on processing of large amount of data generated by sensors (e.g., laser, video, etc.) [2]. Either due to constraints of computation and battery resources on the end devices or due to coordination requirements, the corresponding sensor data is transmitted to a powerful server for processing in order to meet the short response time requirements [3]. *Edge Computing* (EC) [4], [5] is meant to cater to these response times as the processing units, known as *Edge Servers* (ESs), are placed close to end devices. These servers have huge processing capacity compared to the end devices and thus offer reduced computation time. Moreover, due its proximity the communication time remains low. Depending on the size of the factories and the number of applications, many ESs are distributed and orchestrated within the factory [6]. Therefore, due to the mobility of the devices involved in the production process (e.g., milkruns, automated guided vehicles, etc.), the corresponding application running on an ES needs to be migrated along with the mobile device (*MD*) to an ES in its proximity. As described by Govindaraj et al. in [2] it is important to avoid any disruptions in the production process. Thus the service corresponding to the *MD* needs to be migrated live such that the *MD* cannot perceive any interruption.

978-1-7281-0568-0/19/$31.00 ©2019 IEEE

Live migration is a process of moving application or an execution environment between different hosts without loss of their runtime state. The execution environment can be a process running on the bare metal [7], a Virtual Machine (VMs) [8] or a container [9]. To improve the understandability, we address the execution environment as application throughout the paper. We aim to support uninterrupted functioning of *MD*s in a distributed EC infrastructure with the help of live migration. However, all the state-of-the-art live migration mechanisms involve a client-perceptible service unavailability which is commonly known as *downtime*. Hence, these migration schemes are not suitable for Industry 4.0 applications with real-time requirements.

Thus, the goal of this work is to design and investigate a live migration scheme with *zero downtime*, i.e no interruption perceivable by the client. We propose a new migration scheme which combines currently available migration scheme with the concept of hot-standby redundancy. To achieve uninterrupted service live migration, we exploit the redundant back-up infrastructure used in the industry for safety and reliability purpose [10]. Redundancy is a common failsafe mechanism used in safety critical applications (e.g., airplane communications), in which the secondary server takes over the operation if the primary server has to fail [11]. A seamless migration involving redundant servers require complicated network management, for which we use Software-Defined Networking (SDN).

We structure this paper as follows: Section II focuses on analyzing currently available live migration schemes. As an improvement over the existing schemes, Section III presents our proposed uninterrupted service live migration scheme and the corresponding design architecture. Subsequently, Section IV shows the evaluation of our algorithm compared to the known best migration scheme. And finally, Section V concludes the paper by presenting an outlook.

II. Related Work

Live migration is a popular process in the cloud computing paradigm to move services between data centers with

minimal downtime [12]. It is mainly beneficial in failover mechanisms. In case a hardware failure is suspected in a server, the running execution system can be live migrated to an alternative server before the service is disrupted [8]. It also enables dynamic load balancing when the server resource such as memory, storage or processing capacity is overloaded, by transferring the execution environment to another server with sufficient resources [13]. Furthermore, live migration can also be helpful in improving the energy efficiency [14] of the computing infrastructure by suspending unused hosts or servers and migrating the running environments to a desired server. Besides that, it improves the fault-tolerance of whole execution environment [15]. For all the stated reasons, live migration is becoming popular and is gaining importance in edge computing. Nevertheless, there are very few works that deal with live migration in the context of edge computing. We focus on these four migration schemes: pre-copy, post-copy, hybrid, and redundancy migration to underscore the need for a novel migration scheme.

A. Pre-Copy Migration

Theimer et al. introduce pre-copy migration in [16] that consists of two phases: push phase and stop-and-copy phase. During the push phase, the source server ($S_{Src.}$) iteratively copies the memory corresponding to the service to the destination server ($S_{Dest.}$). Eventually $S_{Src.}$ covers the whole memory corresponding to this application, known as Writable Working Set (WWS) [8]. However, during this phase $S_{Src.}$ continues to run the application thus constantly modifying its memory pages. $S_{Src.}$ now copies only the delta of the memory to $S_{Dest.}$ iteratively. This continues until the number of altered memory pages to be transferred reduces to a certain size or the number of iterations exceeds a certain threshold. Now, $S_{Src.}$ briefly suspends the application, called as stop-and-copy phase, to copy its consistent state to $S_{Dest.}$. On receiving the the complete state of the application, $S_{Dest.}$ restores the application and resumes it successfully. During this phase the client experiences a service downtime.

There has been many efforts to optimize the downtime and the migration time in the past. Sharma er al. [17] introduce a three phase optimization framework for pre-copy migration. They propose a heuristic approach for the selection of the memory pages to be transferred in the first phase in the WWS. Then the migration controller executes the prediction algorithm to classify frequently and non-frequently updated pages in order to reduce the transfer of duplicate pages during the second phase. Since the downtime in pre-copy depends on the size of the consistent state to be transferred in the last iteration, the authors try to reduce the size of the data to be transmitted by proposing a Run-Length Encoding (RLE) compression technique. They claim to reduce the migration by 70% and downtime by 3% compared to traditional pre-copy migration. However, the WWS can be sufficiently large depending on the type of the applications running [18]. So, the RLE compression technique can also not make much effect in reducing the downtime for applications with large state sizes.

B. Post-Copy Migration

Unlike pre-copy migration, Zayas [19] proposes post-copy migration as a concept of copy-on-reference. This scheme consists of stop-and-copy phase and pull phase. During the stop-and-copy phase, $S_{Src.}$ freezes the service to transfer only the CPU state to $S_{Dest.}$ and the client experiences downtime. In his migration scheme, Zayas reduces the data transfer during the stop-and-copy phase to the essential CPU state as most applications only use a little fraction of their total allocated memory WWS. On receiving the CPU state, $S_{Dest.}$ restores and resumes the application. However, when it starts to serve the client, it experiences pagefaults on every memory page request as it is not yet transferred from $S_{Src.}$. Too many pagefaults can lead to degraded performance of the application.

Though post-copy has a lower downtime compared to the pre-copy scheme, the solution suffers from inconsistent performance during the pull phase and increases the migration-time tremendously based of the characteristics of the application and number of pagefaults. To overcome the inconsistency during the pull phase, Hines et al. propose an optimization of the traditional post-copy migration [20] called pre-paging. Pre-paging refers to a mechanism of predicting pages which will be accessed by the destination. By predicting them in advanced, $S_{Dest.}$ can request them from the $S_{Src.}$ even before the page fault occurs. However, pre-paging is not a deterministic process, rather a stochastic approach, which can lead to some unwanted scenarios. Thus, this scheme is also not suitable for the Industry 4.0 applications.

C. Hybrid Migration

Pre-copy and post-copy migration have complementing performances. Combining both the schemes results in a good trade-off and is called hybrid migration [21]. Thus, hybrid migration consists of three phases: push phase, stop-and-copy, and pull phase. During the push phase, $S_{Src.}$ transfers an image of all the states to $S_{Dest.}$ as in pre-copy migration scheme. The duration of the push phase is also an optimization parameter depending on the type of application. So the end time of push phase is relative. As a next step, $S_{Src.}$ executes the stop-and-copy phase which leads to a service downtime as in previous schemes. The resumable copy of the environment can be any of post-copy or pre-copy variant, e.g., only a minimal state transfer or CPU context including a consistent copy of the working set, identified via a heuristic. Finally, $S_{Dest.}$ restores the application and requests the missing memory pages during the pull phase. Depending on the amount of missing information at the destination, the duration of pull phase varies. Hybrid migration is basically an optimized migration scheme with a lot of possible optimization parameters.

Hybrid migration can optimize the downtime as well as migration time since the implementation allows many parameters for optimization [21]. Lei et al. [22] use a markov model to predict the memory page access to reduce pagefaults in the pull phase. The existing work shows many attempts to reduce the downtime but does not provide a robust live migration including fault tolerance and seamless handover.

D. Redundancy Migration

To reduce the downtime experienced due to stop-and-copy phase in the other migration schemes that is caused by creation, transmission, and restoration of a consistent state image on the destination server, we propose a novel scheme called *Redundancy Migration* in [23]. We have evaluated this scheme for linux containers, but it is application agnostic and environment independent. Migration controller and traffic controller installed on $S_{Src.}$ and $S_{Dest.}$ assist the migration process. When the system triggers a migration, the client starts to communicate with $S_{Dest.}$ and the migration controller on $S_{Dest.}$ starts the buffer and routing initialization phase. During this phase $S_{Src.}$ continues to serve the client while the traffic controller on $S_{Dest.}$ buffers the incoming packets and simultaneously forwards to $S_{Src.}$. Meanwhile, the migration controller creates a checkpoint of the consistent state of CPU, network, and memory on $S_{Src.}$ during which a small downtime is observed. $S_{Dest.}$ on receiving this checkpoint, restores and resumes the application. However, since the client continues to communicate with $S_{Src.}$, the WWS continues to change. Thus, in order to synchronize the states, $S_{Dest.}$ replays the packets stored in the buffer. $S_{Dest.}$ will eventually catch up to the same state on $S_{Src.}$. It is important to note that this scheme works if the packet processing speed on the destination server is faster than the packet generation rate on the client. Nevertheless, there is a brief interruption in the checkpoint creation time. Depending on the size of the application state, the client experiences a brief downtime.

Thus, to the best of our knowledge, none of the existing approaches can guarantee an uninterrupted service to the clients during live migration. Considering redundancy migration as a benchmark, we present an uninterrupted hot-standby migration scheme in the following section. It is important to notice here that an interruption for our application is asssumed by a Round Trip Time (RTT) over 10 ms.

III. Uninterrupted Hot-standby Migration

As mentioned in Section I, the Industry 4.0 applications require short response times and high availability. Thus, the goal of this migration scheme is to provide a seamless service handover and thus an uninterrupted service to the client throughout its operation. This in turn implies that the client does not experience an RTT more than its threshold set.

We base our scheme on the redundancy migration scheme but combine it with the concept of redundant servers [11]. Since we propose this as a solution for live migration in edge computing infrastructure, we consider two edge servers in the vicinity as primary and secondary source servers (*primary-$ES_{Src.}$* and *secondary-$ES_{Src.}$*) and similarly primary and secondary destination servers (*primary-$ES_{Dest.}$* and *secondary-$ES_{Dest.}$*). Furthermore, we use SDN switches to manage the flows between the client and the servers. An SDN controller controls the switches and instructs them to forward, duplicate or drop the packets based on the decisions made by the edge controller.

Fig. 1 illustrates an exemplary setting of a client and edge servers and controller with SDN switches and controller. Initially the client communicates with the source edge servers, *primary-$ES_{Src.}$* and *secondary-$ES_{Src.}$*. We represent the *SDN controller* and an *edge controller* as two different entities. The *edge controller* decides when the client service needs to be migrated from source to destination edge servers and the *SDN controller* directs the flows accordingly. In our uninterrupted redundancy migration scheme, we propose five phases to achieve *zero downtime* as shown in Fig. 2.

Initial phase: Initially the *MD* is connected to *primary-$ES_{Src.}$* and *secondary-$ES_{Src.}$* simultaneously over an infrastructure as shown in Fig. 1. Same application runs on both *primary-$ES_{Src.}$* and *secondary-$ES_{Src.}$* in a hot-standby redundancy manner. *MD* initially tries to communicate with *primary-$ES_{Src.}$* via *SDN-switch$_1$*. However, on receiving the packet from *MD*, *SDN-switch$_1$* queries *SDN controller* for the action that it needs to follow. Based on the command of *SDN controller*, *SDN-switch$_1$* replicates the packet to *secondary-$ES_{Src.}$*. *primary-$ES_{Src.}$* and *secondary-$ES_{Src.}$* process the packets independently and reply via *SDN-switch$_1$*. By default, *SDN-switch$_1$* forwards the reply sent by *primary-$ES_{Src.}$* unless until specified otherwise.

Handover phase: We consider here an *MD* moves across the factory floor in a predefined route. Based on the coverage area of a serving an access point (AP), the client undergoes a communication handover to a neighboring AP. We consider this as a trigger for migration. The client now starts to communicate via *SDN-switch$_2$* via AP_3. *SDN-switch$_2$* forwards the first packet to *SDN controller* for the routing decision. Since *primary-$ES_{Src.}$* and *secondary-$ES_{Src.}$* are still serving, *SDN controller* instructs *SDN-switch$_2$* to forward the packets accordingly. As *edge controller* has triggered the migration, it initiates the migration phase.

Migration phase: Simultaneously the *edge controller* instructs a snapshot creation on *secondary-$ES_{Src.}$* of the application which constitutes a consistent state of CPU, network, and memory. Meanwhile, *primary-$ES_{Src.}$* continues to serve the *MD* uninterruptedly. On receiving the state, the destination servers *primary-$ES_{Dest.}$* and *secondary-$ES_{Dest.}$* restores the execution environment. Meanwhile, from the instance of snapshot creation, the incoming packets from the *MD* are continuously buffered on *SDN-switch$_2$*. Once the applications are resumed on *primary-$ES_{Dest.}$* and *secondary-$ES_{Dest.}$*, they replay the buffered packets in order to catch up the current state on *primary-$ES_{Src.}$*. However, we assume that the rate of processing the packets from the buffer is faster than the rate at which it is filled, i.e., the packet generation rate of the *MD*.

Review phase: At this instance, the buffer is empty and both *primary-$ES_{Dest.}$* and *secondary-$ES_{Dest.}$* should have the same application state as *primary-$ES_{Src.}$* and *primary-$ES_{Dest.}$*. However, we review that by comparing the output. *Edge controller* instructs the *SDN controller* to forward the responses from all the four servers to it to verify the state consistency. Until the decision is made *primary-$ES_{Src.}$* continues to serve the client. *edge controller* initiates a counter for N consecutive

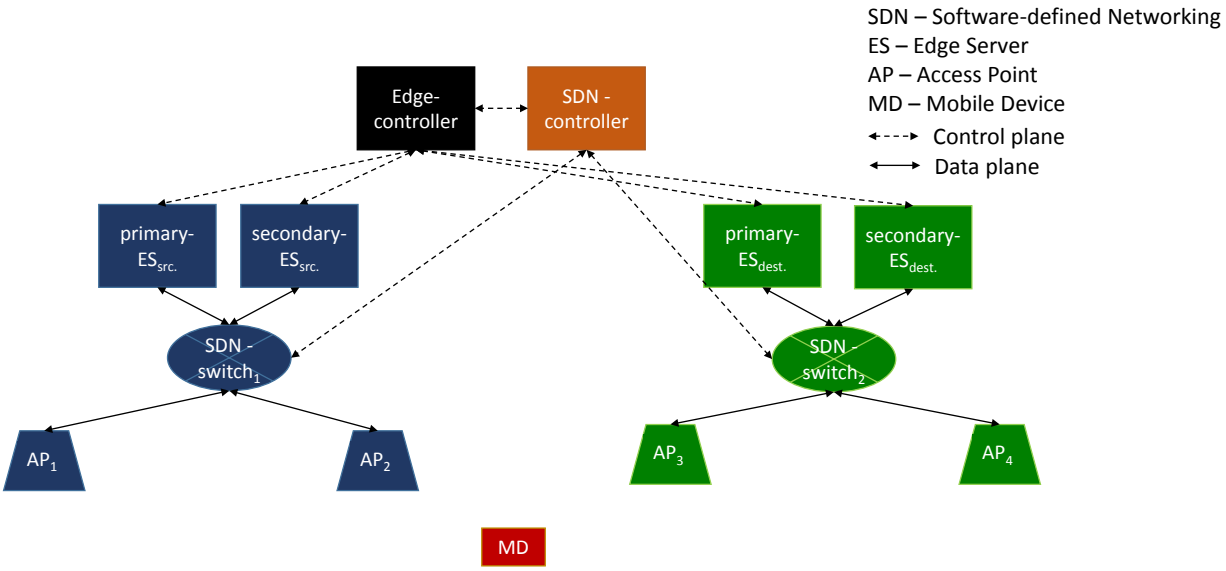

Fig. 1: Overview of the most relevant part of our setup: It consists of identical applications running on redundant source edge servers. Client communicates with *primary-ES*$_{Src.}$ and *secondary-ES*$_{Src.}$ via SDN-switch. After the live migration the client connects to redundant destination edge servers, *primary-ES*$_{Dest.}$ and *primary-ES*$_{Dest.}$

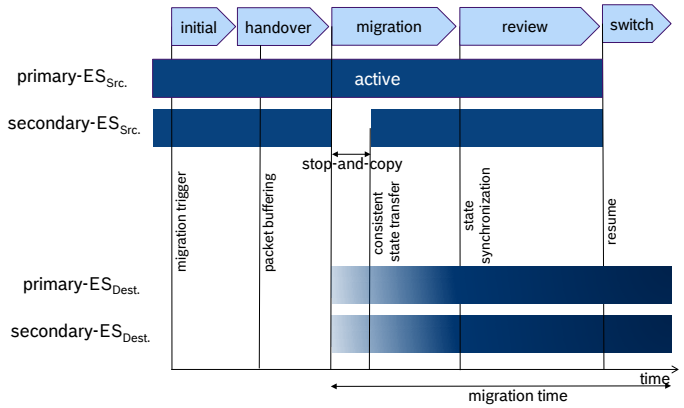

Fig. 2: Uninterrupted hot-standby migration phases: Our setup consists of identical MEC applications running on redundant source Edge Servers. *MD* communicates to the *primary-ES*$_{Src.}$ via *SDN-switch*$_1$ and the *SDN-switch*$_1$ replicates the packets to *secondary-ES*$_{Src.}$ as well. After the live migration, the *MD* connects to two new redundant edge servers, *primary-ES*$_{Dest.}$ and *primary-ES*$_{Dest.}$.

matches of the responses from *primary-ES*$_{Src.}$, *primary-ES*$_{Dest.}$, and *secondary-ES*$_{Dest.}$. The process repeats until the condition is met.

Switch phase: On verifying N matches, *edge controller* confirms that *primary-ES*$_{Dest.}$ and *secondary-ES*$_{Dest.}$ are synchronized with *primary-ES*$_{Src.}$ and can now take over the service. At this instance, the *SDN controller* instructs *SDN-switch*$_2$ to forwards the incoming packets from *MD* only to *primary-ES*$_{Dest.}$ and *primary-ES*$_{Dest.}$. This marks the completion of migration process.

During the whole process of migration the client was always served by the *primary-ES*$_{Src.}$ until the switch-over happened to *primary-ES*$_{Dest.}$ and *primary-ES*$_{Dest.}$. With redundant servers, though the resource overhead is high, it provides an uninterrupted service to *MD* thus ensuring low response time and high availability requirements of Industry 4.0 applications.

We implement this migration scheme as described above using NS-3 simulator to verify its performance and compare it against redundancy migration.

TABLE I: Simulation campaign for the NS-3 evaluation setup

Parameter Name	Values
Model Parameter	
Tx Interval	f(x) = NormalRandomVariable (μ,σ)
Snapshot time	f(x) = NormalRandomVariable (μ,σ)
Restoration time	f(x) = NormalRandomVariable (μ,σ)
Replay time	f(x) = NormalRandomVariable (μ,σ)
Fixed Parameter	
Wireless channel	5 GHz (IEEE 802.11a)
Ethernet channel	1 Gbps
Delay	0.2 μs
Packet size	624 bytes
Server processing delay	1 ms
client mobility	Linear
Ranges Parameter	
Application size	x = {4, 20, 40, 2000} MB
Repetition	
Simulation number	100 times

IV. Evaluation Results

We have made a realistic implementation of the factory network using NS-3 simulator. It consists of a tree topology with realistic link and edge server capacities as well as mobile devices with realistic application behavior. Table I shows the simulation campaign or the parameter values we have considered for our simulation setup. Since redundancy migration is considered to be the best method to reduce the downtime to its lowest possible, we compare our algorithm with it.

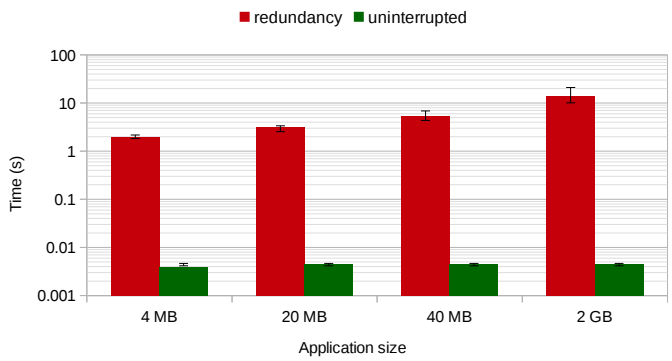

Fig. 3: Comparison of downtime between redundancy migration and uninterrupted migration. The plots are provided for migration of different application size.

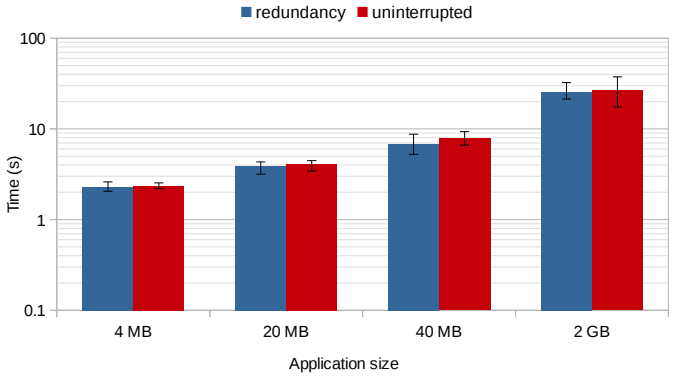

Fig. 4: Comparison of overall migration time between redundancy migration and uninterrupted migration. The plots are provided for migration of different application size.

As shown in Table I, the packet generation rate of an application is realized as a normal distribution model `NormalRandomVariable` with mean value μ = 10 ms and variance σ = 0.2 ms. However, as mentioned in Section III, the buffer processing rate has to be maintained faster than the packet generation rate. Hence, we model this replay time as a Normal distribution model `NormalRandomVariable` with mean time μ = 18 μs and variance σ = 40 μs. Other model parameters are snapshot creation time and restoration time. However, these values are application specific depending on if it is CPU intensive or memory intensive. As described in Section III, we model the application snapshot creation at *primary-ES*$_{Src.}$ and the restoration on *primary-ES*$_{Dest.}$ and *secondary-ES*$_{Dest.}$. To ensure a realistic behavior, we consider the container snapshot and restoration values presented in our earlier work [23]. The application state size simulated ranges from 4 MB, that may correspond to a sensor data processing application, to 2 GB, that may correspond to an augmented reality application.

With an objective to keep the RTT experienced by an *MD* low, we trigger the migration as and when the RTT goes below the set threshold value. Fig. 3 shows a comparison of the resulting downtime in redundancy migration and our uninterrupted migration scheme represented in logarithmic scale for different application state size.

It clearly shows that, the downtime in redundancy migration increases with the increasing state size whereas, in uninterrupted migration the *MD* experiences a constant downtime of approx. 4.6 ms irrespective of the application state size. Therefore, the *MD* experiences the same downtime for any application. This downtime in uninterrupted migration is due to the time involved in decision making process in the *SDN controller*.

On the other hand, Fig. 4 shows the comparison of migration times represented in logarithmic scale for different application state size. It clearly indicates that the migration time increases with increasing state size, which is due to the longer packet replay time involved in the review phase. Though the migration time experienced by the *MD* is similar in both the schemes, it is worth noting that the uninterrupted migration requires additional resources and causes additional overhead. Nevertheless, the extra price is to achieve a robust, fault-tolerant, and seamless live migration.

V. Conclusion and Outlook

In this paper, we presented a novel live migration scheme called uninterrupted hot-standby migration that ensures an uninterrupted service to the mobile devices in Industry 4.0 scenario. We exploit the redundant back-up infrastructure used in the industry for safety and reliability purposes in order to achieve a robust migration. Irrespective of the application size, the downtime remains constant (few ms) thus supporting a seamless migration. On the downside, our approach introduces network overhead and redundant use of multiple servers. However, this is limited to a short duration of the migration. Also, it is worth noting that our migration is rather relevant for applications that can tolerate downtime in milliseconds.

As an extension of this work, we want to explore the limitations of our approach in the real world implementations using off-the-shelf SDN and Edge Computing infrastructure. This involves challenges such as virtualization of an industrial application, enabling seamless communication handover, provisioning of edge resources and many more. We aim to investigate the overall quality of service experienced by various devices on the factory floor during the process of migration.

Acknowledgment

This work has been supported by the Federal Ministry for Economic Affairs and Energy of the Federal Republic of Germany (Foerderkennzeichen 01MA17008D, IC4F). The authors alone are responsible for the content of the paper.

References

[1] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer Networks*, vol. 101, no. Supplement C, pp. 158 – 168, 2016, industrial Technologies and Applications for the Internet of Things. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615005046

[2] K. Govindaraj, A. Artemenko, D. Grewe, and A. Kirstdter, "Towards zero factory downtime: Edge computing and sdn as enabling technologies," in *The International Workshop on Mobile Edge Networks and Systems for Immersive Computing and IoT*, 2018.

[3] J. H. Lala and R. E. Harper, "Architectural principles for safety-critical real-time applications," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 25–40, Jan 1994.

[4] N. Fernando, S. W. Loke, and J. W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Comp. Syst.*, vol. 29, pp. 84–106, 2013.

[5] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.

[6] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, "How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective," *International journal of mechanical, aerospace, industrial and mechatronics engineering*, vol. 8, no. 1, pp. 37–44, 2014.

[7] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration," *ACM Comput. Surv.*, vol. 32, no. 3, pp. 241–299, Sep. 2000. [Online]. Available: http://doi.acm.org/10.1145/367701.367728

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251223

[9] Y. Qiu, C. H. Lung, S. Ajila, and P. Srivastava, "Lxc container migration in cloudlets under multipath tcp," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, July 2017, pp. 31–36.

[10] I. E. Commission *et al.*, "Functional safety of electrical/electronic/programmable electronic safety related systems," Tech. Rep., 2000.

[11] A. Höller, N. Kajtazovic, C. Preschern, and C. Kreiner, "Formal fault tolerance analysis of algorithms for redundant systems in early design stages," in *Software Engineering for Resilient Systems*, I. Majzik and M. Vieira, Eds. Cham: Springer International Publishing, 2014, pp. 71–85.

[12] R. C. T. Erl and A. Naserpour, *Cloud Computing Design Patterns*. Upper Saddle River, NJ, USA: Prentice Hall Press, 2015.

[13] B. Gerofi, H. Fujita, and Y. Ishikawa, "An efficient process live migration mechanism for load balanced distributed virtual environments," in *2010 IEEE International Conference on Cluster Computing*, Sept 2010, pp. 197–206.

[14] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "Enacloud: An energy-saving application live placement approach for cloud computing environments," pp. 17–24, Sept 2009.

[15] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization," pp. 23–32, 2007. [Online]. Available: http://doi.acm.org/10.1145/1274971.1274978

[16] M. M. Theimer, K. A. Lantz, and D. R. Cheriton, "Preemptable remote execution facilities for the v-system," *SIGOPS Oper. Syst. Rev.*, vol. 19, no. 5, pp. 2–12, Dec. 1985. [Online]. Available: http://doi.acm.org/10.1145/323627.323629

[17] S. Sharma and M. Chawla, "A three phase optimization method for precopy based vm live migration," *SpringerPlus*, vol. 5, no. 1, pp. 1–24, 2016.

[18] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation." *CloudCom*, vol. 9, pp. 254–265, 2009.

[19] E. Zayas, "Attacking the process migration bottleneck," *SIGOPS Oper. Syst. Rev.*, vol. 21, no. 5, pp. 13–24, Nov. 1987. [Online]. Available: http://doi.acm.org/10.1145/37499.37503

[20] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '09. New York, NY, USA: ACM, 2009, pp. 51–60. [Online]. Available: http://doi.acm.org/10.1145/1508293.1508301

[21] S. Sahni and V. Varma, "A hybrid approach to live migration of virtual machines," in *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Oct 2012, pp. 1–5.

[22] Z. Lei, E. Sun, S. Chen, J. Wu, and W. Shen, "A novel hybrid-copy algorithm for live migration of virtual machine," *Future Internet*, vol. 9, no. 3, 2017.

[23] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *23rd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2018, Torino, Italy*, Sept 2018, pp. 83–90. [Online]. Available: https://doi.org/10.1109/ETFA.2018.8502659