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 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)

Smart Resource Planning for Live Migration in Edge Computing for Industrial Scenario

Keerthana Govindaraj*, Jibin P John†*, Alexander Artemenko*, Andreas Kirstaedter‡
*Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany
{keerthana.govindaraj, alexander.artemenko}@de.bosch.com
†University of Bremen, Institute of Communication Networks and Computer Engineering, Bremen, Germany
{jibin.john}@uni-bremen.de
‡University of Stuttgart, Institute of Communication Networks and Computer Engineering, Stuttgart, Germany
{andreas.kirstaedter}@ikr.uni-stuttgart.de

***Abstract*—Future factory automation systems are expected to process vast amounts of data and orchestrate complex cyber-physical components. This involves devices such as autonomous guided vehicles and augmented reality glasses that are inherently mobile. Edge Computing (EC) is a promising approach to address the requirements set by upcoming industrial systems. However, the combination of distributed edge network and multiple mobile devices gives rise to the need for live migration. Migration of multiple services in parallel demands for an intelligent allocation of network resources. This paper describes a dynamic algorithm to perform smart allocation of resources for live migration on demand. In addition to maintaining the Quality of Service requirements of the end devices, it also maintains the network overhead due to live migration low. We evaluate the performance of our algorithm using a discrete-event network simulator.**

***Index Terms*—factory automation, edge computing, live migration, scheduling**

I. Introduction

In the scope of *Industry 4.0*, the demand to interconnect millions of sensors and actuators is increasing. The ultimate goal is to automate the factories and make it reconfigurable on-demand by linking multiple entities on the factory floor. However, the devices involved in factory automation have unique requirements such as low end-to-end latencies, processing of huge data, coordination of various cyber-physical systems, etc. Most of the existing devices are equipped with a dedicated hardware that experiences constraints on the battery capacity and the processing power. They are also inflexible and cannot react to varying demands. Thus, offloading the applications of these devices to a virtualized environment on a powerful server resolves these problems [1].

Edge Computing paradigm has already paved its way into factories to address the stringent requirements of applications such as smart analytics, big data, augmented reality, etc. [2]. According to this paradigm, these powerful computational entities known as *Edge Servers* (ESs) are placed close to the devices. Thus, the resource constrained end devices can profit by offloading their applications to these ESs and experience the round trip time (RTT) within the critical limit.

Fig. 1 shows an exemplary use case in which the factory automation system involves various static and mobile devices (MDs). Devices such as cameras and LIDARs continuously generate huge amount of data that need to be processed in order to coordinate the functioning of different devices. MDs such as autonomous guided vehicles [3] are also becoming popular in automating transportation tasks and are inherently mobile, however, in a planned route. Additionally, humans using augmented reality glasses [4] and mobile control panels [5] move around in a random fashion. All these devices have different kinds of applications, memory or computation intensive, varying mobility-speed as well as different soft and hard RTT thresholds.

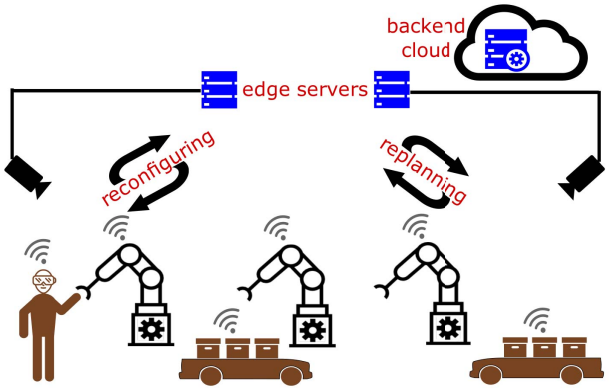

Fig. 1. Illustration of a factory automation use case with static and mobile devices in an edge computing infrastructure.

We consider that with the advent of 5G technology in the industrial environment, many of these end devices, mainly the MDs will be connected in a wireless fashion to the backend network [6]. This provides huge data throughput, low communication latencies as well as high flexibility to the end devices. Thus, with the increasing number of end devices, the factory backbone network with the existing capacity will become the actual bottleneck.

Therefore, as shown in Fig. 2 it is ideal to place the ESs on different levels of tree topology of the factory network. We term the ESs that co-exist with access points as ES_{level1}, the ones with access switches as ES_{level2}, the ones with distribution switches as ES_{level3} and finally the ones with core switches as ES_{level4}. ESs on different levels cater to various RTT depending on the load on network and ES.

2573-7562/19/$31.00 ©2019 IEEE
DOI 10.1109/MobileCloud.2019.00012

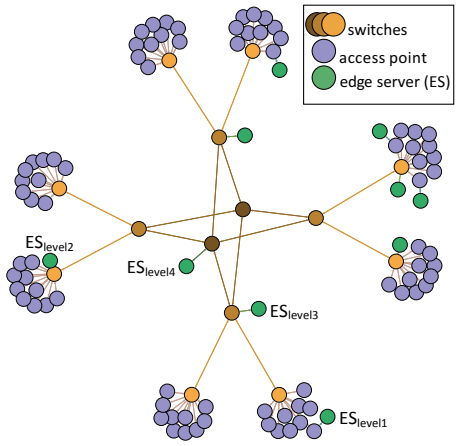

Fig. 2. Exemplary illustration of the industrial network topology and placement of edge servers in the network.

However, a *Mobile Device Application* (MDA) offloaded on ES_{level1} experiences frequent handovers depending on its mobility, whereas on ES_{level4} it may experience increased RTT depending on the number of end devices communicating with ES_{level4}. Thus, it is ideal to offload an MDA either on ES_{level2} or ES_{level3} depending on the latency requirement. Nevertheless, with the mobility, the MDA has to communicate with the ES over multiple hops which again leads to increased RTT. Therefore, the MDA must be migrated to an ES in its vicinity to maintain the expected RTT. Furthermore, the service downtime of the MDA need to be within the critical limits also to maintain the expected RTT and to avoid any disruption in the system.

In our previous work, we proposed a novel live migration scheme known as redundancy migration [7], that on one hand reduces the downtime but on the other hand leads to increased migration time. Increased migration time implies redundant use of resources for a longer time which in turn makes the migration process expensive. Eventually, increasing number of MDAs will lead to parallel migrations which in turn further overloads the backend network. Hence, we need to schedule these parallel migrations such that the devices do not experience increased RTT, do not overload the network, and maintain the redundant usage of ES resources as low as possible. Furthermore, the number of migrations need to be kept low as it is an expensive process.

To achieve this, we firstly examine the various requirements of live migration process and MDAs to characterize them. Then, by combining the well known techniques in a novel manner, we achieve an optimized scheduling scheme for service live migration of MDAs. We validate our algorithm for its performance and quality using the NS3 simulation environment.

II. Related Work

A. Live Migration

Live migration is the process of moving an environment from one server to another without losing the state of the application running, independent of the virtualization environment. The execution environments can be a Virtual Machine (VM) [8], a container [7], or a process [9]. This involves migration of CPU state, memory state, and network state. On the destination server, the CPU, memory, and network states are required in order to restore the service correctly.

CPU and Memory State Migration: Different live migration schemes exist in the literature, pre-copy [10], post-copy [11] and hybrid [12]. The process of live migration is measured by two important metrics, namely downtime and migration time. Downtime corresponds to the duration in the migration process during which the service is unavailable to the device. During this period, the user can sense the interruption in service if the downtime is greater than the Quality of Service (QoS) requirement of the application. Therefore, downtime is an important metric as it influences the service availability factor and Service Level Agreement (SLA). Migration time corresponds to the complete duration since the trigger of migration until the service is restored on the destination server and resumed. At this point the source server can shut down the service and no further actions need to be taken place. During this period, the resources are occupied on both the hosts along with network resources and thus the migration trigger needs to be planned and scheduled. Clark et al. [8] show that live migration consumes significant bandwidth for several seconds. The relevant work includes many proposals to optimize the existing migration schemes to either improve the downtime or the migration time [13]–[16]. Nevertheless, none of the existing migration schemes fulfil the downtime requirements of the industrial applications [2].

Network management: Most of the work on live migration concentrate on CPU and memory state migration. However, when the service needs to be migrated between different subnets, layer 3 network protocol causes the connection between the device and the server to fail. The service initiation time on the new device-server leads to downtime in the service. Especially, in a distributed edge network as shown in Fig. 2, an MD experiences a poor QoS due to multiple migrations when changing location. NiranjanMysore et al. [17] extend the layer 2 routing and forwarding protocol for data center environments. However with the increasing size of the topology and the number of server, this method is not viable. Other popular methods are Virtual Private Network (VPN) [18], Multipath TCP [19], mobile IP [20] and IP tunnels [21]. However, all these methods add an additional layer of management for a seamless connection migration. Unlike the traditional methods, Benet et al. [22] make use of Software-Defined Networking (SDN) and OpenFlow methods to achieve fast restoration of network connectivity after the migration.

Thus, in order to meet the downtime requirements, we consider the redundancy migration scheme proposed in [7] to transmit the CPU and memory states and an SDN based approach [22] for the network management.

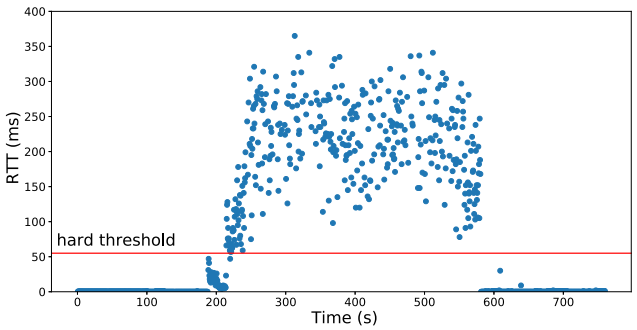

Fig. 3. RTT between a mobile device and its corresponding edge server on a circular path

B. Scheduling

Scheduling schemes are very popular in various fields. However, we focus on the schemes existing in the field of computer and communication network theory. The most popular ones are first come, first served; shortest job first; priority-based scheduling; round robin; multilevel queue scheduling; earliest deadline first; etc. Some novel approaches are proposed in the distributed systems. Wang et al. [23] propose a utility-based scheduling scheme that focuses on bulk data transfers between the distributed system. They use the time-utility function to assign the priority to the jobs. However, the algorithm works in a greedy fashion which is not suitable for our purpose. Similarly, Wu et a. [24] propose a scheduling scheme to transfer data between geographically distributed data centers. They rely on SDN to coordinate data transfers by reserving the bandwidth for high priority jobs. But for the purpose of live migration, multiple parameters need to be considered simultaneously in order to fulfill the RTT requirements of the devices. In their vision paper, Stage et al. [25] propose a schedule-based resource provisioning for VMs. The authors aim to reduce the overall server costs as well as avoid the network congestion. Similarly, Balman [26] tries to resolve the resource conflict while shifting bulk data from one data center to another. Also here, the author aims to avoid network sharing between two migration jobs but does not consider the varying bandwidth. Veeraraghavan et al. [27] propose a heuristic to address the problem due to dynamically varying bandwidth. Nevertheless, to the best of our knowledge, none of the existing algorithms in the literature meant for scheduling service live migration simultaneously consider the mobility of devices, corresponding RTT requirements, varying network bandwidth, and priority.

III. Smart Scheduling Algorithm for Parallel Migrations

We consider a tree topology as shown in Fig. 2 but take only the $\text{ESs}_{\text{level2}}$ into consideration for scheduling. For the sake of simplicity, we assume that the MDs are moving at constant velocity, the direction of motion is predetermined, and the MDs can be exactly localized at any given instant of time. However, to make it more realistic, various accelerations and directions of motion of the MD can be considered with a slight modification in the algorithm. Each end device that has offloaded its application to an ES experiences an RTT, that constitutes the two way communication time and the computation time:

$$RTT = t_{MD-ES} + t_{ESprocessing} + t_{ES-MD}, \quad (1)$$

here we consider t_{MD-ES} as the time required by a packet generated on MD to reach its ES; $t_{ESprocessing}$ as the time required by the ES to process this packet and generate the corresponding response; t_{ES-MD} as the time required by this generated packet on ES to reach the corresponding MD.

Fig. 3 shows the RTT experienced by an MD that starts to move away from its serving ES location at time $0s$ and makes a U-turn to return to the point where it started at time $780s$. The graph clearly shows that with the increasing distance, the RTT experienced rises and decreases as the device returns. Other MDs in the network also experience an increased RTT due to the congestion in the network.

Thus, the goal of this algorithm is to provide a practical and efficient scheduling of live migration processes in order to maintain the RTT of each device within its corresponding threshold. Since live migration is an expensive process, the additional goal is to reduce the number of migration triggers as well as to keep the network load due to migration low.

All the ESs are managed by a controller placed centrally and it continuously tries to keep the network optimized by executing the algorithm. A possible location for the controller can be the factory cloud which has an overview of the complete system.

The controller executes the algorithm in three phases, namely intelligence gathering, job creation and scheduling.

A. Intelligence Gathering

At any given time, the controller has an updated information of the network topology and it keeps refreshing every time something changes. The ESs probe the network at regular intervals by sending an Internet Control Message Protocol (ICMP) packet to estimate the next RTT expected on each link. We use an exponentially weighted moving average using a TCP-like scheme to estimate the RTT:

$$RTT_{est.} = (1-\alpha) * RTT_{avg.} + \alpha * RTT_{obs.}, \ 0 \leq \alpha \leq 1 \quad (2)$$

where α reflects a trade-off between stability and responsiveness and has to be set accordingly to reduce the fluctuations between $RTT_{avg.}$ and $RTT_{obs.}$, the average and observed RTT. Each device has a soft and a hard threshold defined. These can be set based on the quality of service (QoS) for each device. The controller maintains a table with RTTs measured between each $\text{ES}_{\text{level2}}$ and each AP in the network. For the sake of simplicity we consider the RTT from the access points to the ESs since the transmission and propagation time between the MD and AP will be comparatively negligible due to 5G. However, 5G in industrial environment have a limited reachability and thus the range of each AP needs to be characterized before hand. In this work we divide the range of each AP as rectangular areas.

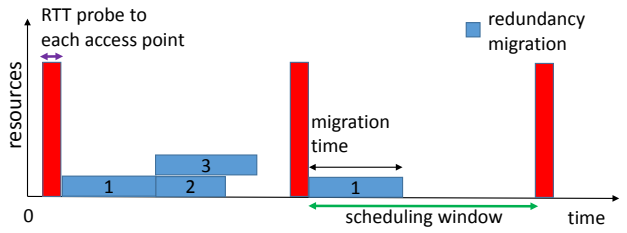

Fig. 4. An illustration of probing, to detect RTT violation and recognize migration jobs, and scheduling windows to sort the jobs in the given window.

The controller also maintains a table with the location of each end device in the network and its corresponding RTT threshold requirements. The controller does a probing in a predefined interval as shown in Fig. 4.

B. Job Creation

The controller creates a migration job for an MDA when it experiences RTT violation. In our algorithm, we combine a reactive and a proactive approach to recognize the RTT violation. And for that, firstly, we need to characterize mobility of the MD.

We model the mobility of each device in an absolute fashion. That is, we do not consider a relative motion between the devices. Thus, an MD moves with velocity v_i. Due to mobility, the MDs constantly undergoes a communication handover. However, we consider it critical only when the device has to communicate over an overloaded communication link and experiences an RTT violation. In the proactive approach, the controller considers the mobility of all the MDs and estimates the distance d_i that MD_i has to move to experience an RTT violation. Thus we obtain the time remaining for MD_i when it needs to communicate with a new ES in its vicinity, and we call it as crucial handover point as at this time instance the migration must be complete in order to avoid RTT violation.

$$t_{handover(i)} = d_i / v_i \tag{3}$$

It is important to note that the algorithm has to be executed in a defined scheduling windows as shown in Fig. 4, such that the network load on the backbone network does not vary during this window.

Secondly, we need to characterize the live migration. We consider redundancy migration [7] as downtime experienced by the industrial applications need to be minimal. The migration time experienced in redundancy migration scheme is higher than in the other traditional methods and is given as:

$$t_{migration} = t_{snapshot} + t_{copy} + t_{restore} + t_{synchronize} \tag{4}$$

$$t_{snapshot} = statesize_{CPU} / speed_{processor}, \tag{5}$$

where CPU state size is given in Bytes and the processor writing speed is given in Bytes per second.

$$t_{copy} = statesize_{CPU+memory+network} / BW_{SD}, \tag{6}$$

where BW_{SD} is the bandwidth available between the source and destination ES.

$$t_{restore} = statesize_{(CPU+memory+network)} / speed_{processor} \tag{7}$$

$$t_{synchronize} = size_{buffer} / speed_{processor}, \tag{8}$$

where $t_{synchronize}$ is the additional time required for state synchronization in redundancy migration. This is highly dependent on how big $size_{buffer}$ is. The type of the application, CPU or memory intensive, influences the $size_{buffer}$ and is directly proportional to the application packet generation rate. Thus, $speed_{processor}$ must be high enough such that the rate at which the buffer is processed is faster that it is filled. Else, $t_{synchronize}$ will be $infinity$. We model the requests arriving at the processor as a Poisson distribution to have a realistic queuing effect experienced by the processor, thus $speed_{processor}$ is realistic.

$$P(x;\mu) = (\exp^{-\mu})(\mu^x)/x!$$

However, we have characterized the values in redundancy migration scheme by repeating the process several times for different applications. Thus, for any given application under full resource availability, the controller can estimate $t_{migration}$.

Based on $t_{handover}$ and $t_{migration}$, the controller calculates the T_{start} for a particular MD. Here, T_{start} is the latest start time at which the migration must be scheduled, or else the device cannot complete its migration before the critical handover occurs and the MD experiences RTT violation. T_{start} is the absolute time calculated by

$$T_{start} = T_{current} + t_{handover} - t_{migration}. \tag{9}$$

Fig. 6 clearly illustrates the scenario when the MD experiences the latest start time. Depending on reactive or proactive approach, the controller recognizes RTT violation and creates a migration job between the source edge server ES_S and the destination edge server ES_D with following attributes

$$Job\{ES_S, ES_D, BW_{SD}, t_{handover}, t_{migration}, t_{start}\}. \tag{10}$$

C. Scheduling

As mentioned in Section I, the factory automation use case consists of several MDs and thus multiple migration jobs will be created. Thus, we schedule the jobs intelligently ensuring that the network is not overloaded. To deal with various scenarios leading to RTT violation, we propose a combination of reactive and proactive approach.

Reactive approach: In this approach, controller creates a job when an MD experiences RTT soft threshold violation. The controller repeats this for all the MDs and creates a list of jobs. For a given application under given conditions, the controller estimates the migration time required based on the history. Next, the controller sorts the jobs in the order of priority. Shortest job gets the highest priority to relieve the congested link at the fastest. Additionally, the controller tries to avoid the resource conflict. Fig. 5 illustrates an exemplary scenario

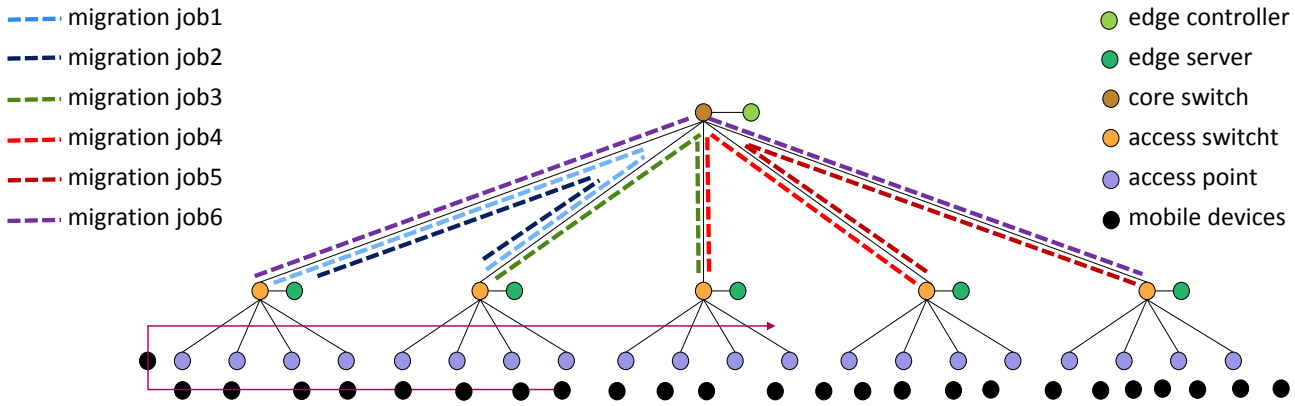

Fig. 5. An exemplary illustration of the jobs arising in the topology on different network links leading to resource sharing.

in which the controller recognizes six jobs with different source and destination ESs. It is obvious that if all six jobs are initiated simultaneously, $job1$, $job5$ and $job6$ share the link thus extending the migration time. Since, in redundancy migration, both source and destination servers need to be kept active throughout the migration time, longer migration time leads to wastage of resources. However, the jobs that do not share the resources are placed simultaneously. Eventually, the controller places the jobs as shown in Fig. 6. All the jobs that do not fit into the window are discarded and controller starts again with the intelligence gathering phase.

Proactive Approach: Unlike the reactive approach, in proactive approach the controller estimates the RTT soft threshold violation in advance based on the mobility of all the MDs and creates a list of jobs. It then uses the information collected for each job in Eq. 10 to sort the jobs in the order of priority. The job with the earliest latest start time T_{start} gets the highest priority. Similar to reactive approach, the controller in proactive approach also tries to avoid the resource conflict as shown in Fig. 5 in order to keep $t_{migration}$ low for all the jobs as shown in. The jobs after schedule are arranged as shown in Fig. 4. However, the window size has to be kept small to avoid jobs targeting to resolve congestion on the same link. Again, all the jobs that do not fit into the window are discarded and controller starts again with the intelligence gathering phase.

In this novel scheduling algorithm, we consider that all the jobs are non-preemptive. This implies that, once the migration is triggered the job cannot be interrupted until its completion. In a given window, we run both reactive and proactive approaches to address various scenarios. If there are *n* jobs to be scheduled, the algorithm requires two iterations to schedule jobs. One iteration is to place the jobs based on the priority and next is to detect any resource conflict. So the complexity of the algorithm is $O(n^2)$.

IV. Simulation Setup

We use NS3 simulation environment to set up a tree topology as shown in the Fig 2. Table I shows the parameters chosen to design the evaluation setup. Since, a complete model of 5G simulation environment is not available, we use the stable

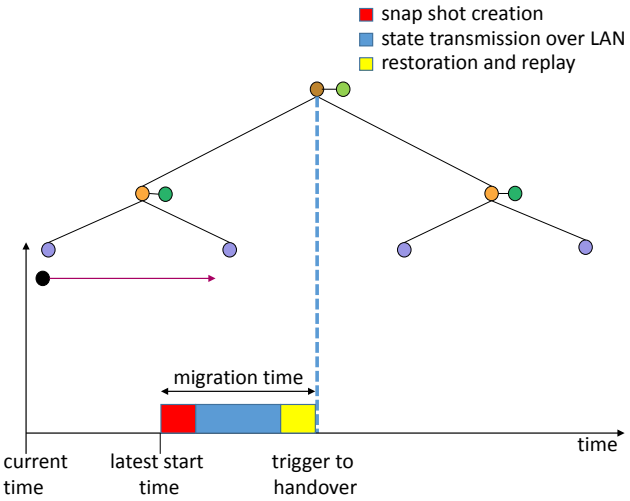

Fig. 6. An illustration of the live migration trigger based on the latest start time to ensure QoS of the mobile device.

model of $IEEE$ 802.11a for the wireless communication. We ensure that the link is not overloaded at any given time such that the bottleneck never arises at the wireless communication link. Certain LAN links are loaded due to the static devices connected at different points. In the initial condition, we have ensured that none of the links are fully loaded. However, since the migration traffic is bursty, it could strongly affect the QoS of all devices in the network. Thus, we consider a dedicated link only for migration data. An equivalent of that can be achieved in real world with the help Virtual Local Area Network (VLAN) [28]. The experiment consists of three different application classes. These applications differ by: packet inter-arrival time; mobility; application state size. Table I provides the details of various applications running in the system.

Most importantly, we consider only the ES_{level2} that co-exist with access switches. All the APs connected as leaflets belong to this ES cluster.

TABLE I
SIMULATION PARAMETERS FOR THE NS-3 EVALUATION SETUP

Parameter	Values
Wireless network	
Protocol	`IEEE 802.11a` (54Mbps)
Range	12 m
Wired network	`IEEE 802.3 Ethernet` (100Mbps)
Simulation time	780 s
No. of repetitions	18
Mobile device	24
Packet size	536 Bytes
Application class A	
mobility	linear; constant speeds (0.5 m/s)
total state size	1000 pages
CPU state size	100 pages
page size	4 KB
packet inter-arrival time	10 ms
soft threshold	15 ms
hard threshold	55 ms
Application class B	
mobility	linear; constant speeds (1 m/s)
total state size	1500 pages
CPU state size	150 pages
page size	4 KB
packet inter-arrival time	5 ms
soft threshold	20 ms
hard threshold	70 ms
Application class C	
mobility	linear; constant speeds (2 m/s)
total state size	1000 pages
CPU state size	100 pages
page size	4 KB
packet inter-arrival time	5 ms
soft threshold	15 ms
hard threshold	55 ms

V. EVALUATIONS AND RESULTS

Using the above mentioned experimental setup we test the performance of our scheduling algorithm. Since the main goal is to ensure the QoS requirements of the devices and the overhead due to migration, it is interesting to compare against a no migration scenario and a location-based migration scenario. The controller in the location-based migration scenario triggers a migration, each time an MD crosses over the cluster boundary. 24 MDs are randomly distributed on the factory floor with each $1/3^{rd}$ of the MDs belonging to a different application class. We repeat the experiment multiple times changing the position of the MDs thus inducing randomness in point of trigger for migration. It is interesting to observe the following effects of using scheduling:

- RTT experienced by all the MDs
- Comparison of number of migration triggers
- Comparison of migrated state size

A. RTT comparison

Fig. 7 shows that the RTT of the MD increases as it moves away from their corresponding edge servers. Thus it is obvious that migration is necessary. It is important to note that the RTT is represented over a logarithmic scale in order to cover the large range. The location based migration however reduces the RTT far below the RTT soft threshold, since the jobs corresponding to all the MDs that cross the ES cluster boundary are blindly migrated. And we notice that the RTT

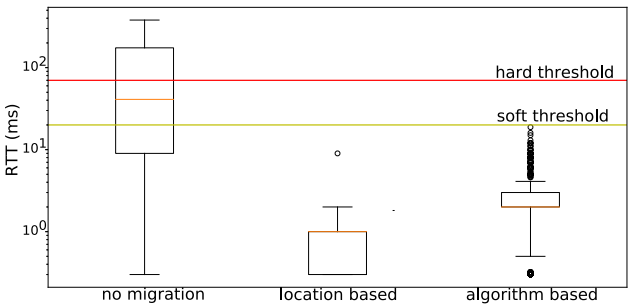

Fig. 7. Comparison of RTT experienced by a mobile device throughout its operation when the application is not migrated at all, when the application is migrated every time the device crosses its location and when the application is migrated based on scheduling

experienced by the MD under our algorithm is higher than in location based migration but still below the RTT soft threshold. Thus we prove that our algorithm is successful in ensuring the QoS requirements expected.

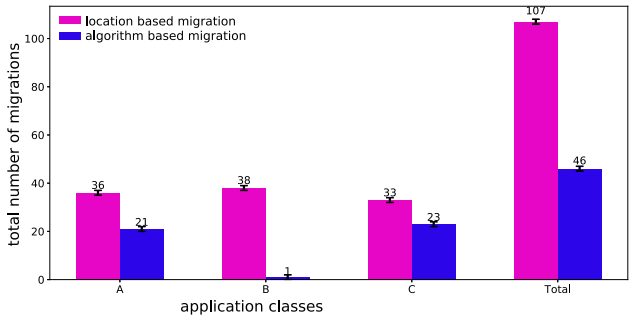

Fig. 8. Comparison of migration triggers in location based migration vs. scheduling algorithm based migration

B. Migration Count

Since resources in the network are limited, migration triggers must be kept very low. Thus we evaluate the performance of our algorithm based on how often the migration was triggered for all the 24 MDAs during 400 repetitions. Fig. 8 shows the number of times the migration triggered for each application class. In the location based migration, since the applications are blindly migrated, there is no priority given to any application. Since our algorithm triggers the migration only when it is required, the number of migrations is comparatively low. We have reduced the migration triggers by 57%.

1) State Size: Similarly, Fig. 9 shows the total state size that has been migrated during the whole simulation for all the 24 MDs in. Since our algorithm prefers the jobs with shorter migration times over the others, the total state size remains low. As seen in Table I, application type 2 has higher state size compared to application type 1 and 3. Therefore, our algorithm chooses application type 2 less often than required. Thus, we successfully reduce the load transmitted by 63%.

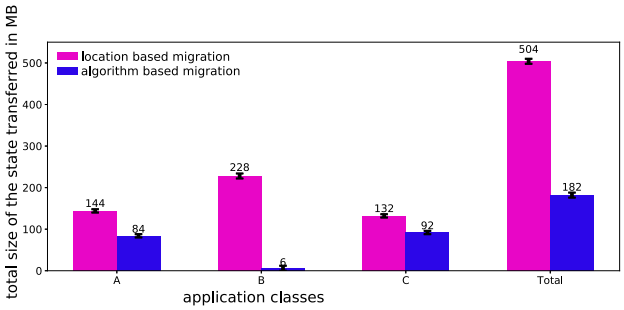

Fig. 9. Comparison of total state size transmitted in location based migration vs. scheduling algorithm based migration

2) Network Overhead: The results clearly show that with our algorithm we can reduce the number of migrations triggered as well as the total state information transmitted. However, it comes at a cost of probing in the network. At the beginning of each scheduling window, every ES_{level2} sends 20 ICMP packets of size 64 bytes each. Thus, 14 kB of ICMP data is sent from a single ES to an AP during the probing interval. The window size here is set to 8 seconds and the algorithm is repeated up to 100 times during this simulation and for all the ESs_{level2}. Hence, probing due to our algorithm leads to data exchange of up to 1.2 MB during the simulation time of 780 s which is considerably low compared to the overall data transmitted in the system.

VI. Conclusion and Outlook

This paper has shown why edge computing is relevant for factory automation and has introduced a use-case with multiple mobile devices with different requirements. The combination of distributed edge computing and mobility gives rise to the need for live migration. Though it reduces the downtime experienced by the system, it is an expensive process in terms of resources utilized. Multiple mobile devices lead to parallel migrations on the same pair of edge servers and the same link. Thus, there is a need for an intelligent algorithm to schedule parallel migration requests such that the round trip times experienced by the devices are always within the expected limits and simultaneously the network load due to migration is kept low. Our algorithm intelligently schedules the migration jobs satisfying all the criteria.

The future work focuses on refining the migration job creation step that can be optimized by improving the job prediction. Also our algorithm relies on the history to estimate the migration time. This can also be further improved by integrating reinforcement learning methods.

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] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, Oct 2009.

[2] K. Govindaraj, D. Grewe, A. Artemenko, and A. Kirstaedter, "Towards zero factory downtime: Edge computing and SDN as enabling technologies," in *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2018, pp. 285–290.

[3] G. Ullrich, *Automated Guided Vehicle Systems: A Primer with Practical Applications*. Springer Publishing Company, Incorporated, 2014.

[4] J. Rambach, A. Pagani, M. Schneider, O. Artemenko, and D. Stricker, "6DoF object tracking based on 3D scans for augmented reality remote live support," *Computers*, Jan 2018.

[5] L. Wang and A. Canedo, "Offloading industrial human-machine interaction tasks to mobile devices and the cloud," in *Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–4.

[6] S. E. Elayoubi, M. Fallgren, P. Spapis, G. Zimmermann, D. Martn-Sacristn, C. Yang, S. Jeux, P. Agyapong, L. Campoy, Y. Qi, and S. Singh, "5G service requirements and operational use cases: Analysis and METIS ii vision," in *European Conference on Networks and Communications (EuCNC)*, June 2016, pp. 158–162.

[7] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2018.

[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 Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286.

[9] 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.

[10] 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.

[11] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, Jul. 2009.

[12] 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.

[13] S. Sharma and M. Chawla, "A three phase optimization method for precopy based VM."

[14] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of the ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '09. ACM, 2009, pp. 101–110.

[15] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, "ReVirt: Enabling intrusion analysis through virtual-machine logging and replay," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 211–224, Dec. 2002.

[16] B. Jiang, B. Ravindran, and C. Kim, *Lightweight Live Migration for High Availability Cluster Service*. Springer Berlin Heidelberg, 2010, pp. 420–434.

[17] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proceedings of the ACM SIGCOMM Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 39–50.

[18] T. Wood, K. K. Ramakrishnan, P. Shenoy, J. V. der Merwe, J. Hwang, G. Liu, and L. Chaufournier, "CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1568–1583, Oct 2015.

[19] F. Teka, C. Lung, and S. Ajila, "Seamless live virtual machine migration with cloudlets and multipath TCP," in *IEEE Annual Computer Software and Applications Conference*, vol. 2, July 2015, pp. 607–616.

[20] S. Kassahun, A. Demessie, and D. Ilie, "A PMIPv6 approach to maintain network connectivity during VM live migration over the internet," in *IEEE International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 64–69.

[21] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the International Conference on Virtual Execution Environments*, ser. VEE '07. New York, NY, USA: ACM, 2007, pp. 169–179.

[22] C. H. Benet, K. A. Noghani, and A. J. Kassler, "Minimizing live VM migration downtime using openflow based resiliency mechanisms," in *IEEE International Conference on Cloud Networking (Cloudnet)*, Oct 2016, pp. 27–32.

[23] X. Wang, W. Tang, R. Kettimuttu, and Z. Lan, "Utility-based scheduling for bulk data transfers between distributed computing facilities," in *International Conference on Parallel Processing Workshops*, Sep. 2015, pp. 175–183.

[24] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. C. M. Lau, "Orchestrating bulk data transfers across geo-distributed datacenters," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 112–125, Jan 2017.

[25] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 9–14.

[26] M. Balman, "Advance resource provisioning in bulk data scheduling," in *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, March 2013, pp. 984–992.

[27] M. Veeraraghavan, H. Lee, E. K. P. Chong, and H. Li, "A varying-bandwidth list scheduling heuristic for file transfers," in *IEEE International Conference on Communications*, vol. 2, June 2004, pp. 1050–1054 Vol.2.

[28] *IEEE 802.1q: VLAN*, IEEE, http://www.ieee802.org/1/pages/802.1Q.html, 2005.