Fog-based Industrial Robotic System: Applications and Challenges

Mohammed Salman Shaik
Václav Struhár, Zeinab Bakhshi
Van-Lan Dao, Nitin Desai
Alessandro V. Papadopoulos
Thomas Nolte
Mälardalen University, Sweden
{name.surname}@mdh.se

Vasileios Karagiannis
Stefan Schulte
TU Wien, Austria
{v.karagiannis, s.schulte}@dsg.tuwien.ac.at

Alexandre Venito
Gerhard Fohler
TU Kaiserslautern, Germany
{venito, fohler}@eit.uni-kl.de

***Abstract*—Fog computing allows to host industrial applications in a cloud-like manner, while providing low latency and on-demand resource availability. Robotics is one industrial domain which stands to benefit from the advantages of fog computing. However, the challenges in developing and implementing a fog-based robotic system are manifold. To illustrate this, in this paper we discuss a system involving robots and robot cells at a factory level, and then highlight the software components necessary for achieving the functionality provided by the robotic systems. Based on this, we provide a simplified reference system architecture and identify key requirements imposed by the architecture to highlight the challenges in implementing such an architecture, with emphasis on resource virtualization, memory interference management, real-time communication, system scalability, dependability and safety. Then, we discuss the challenges from a system perspective where all these aspects are interrelated.**

I. Introduction

Industrial robots are widely used in different automation applications such as for painting and welding in automotive facilities and for packaging in the food industry [1]. More recently, the domain of robotics has evolved to support warehouse automation with mobile robots and, at the same time, emphasis on collaborative robots has also gained significant attention [2]. Legacy software architectures supporting robotic systems needs a significant increase in capabilities when it comes to computational power and flexibility to successfully manage the demands of new applications as well as the need for better inter-operability between machines, and continuous improvements in the performance of individual machines. When evolving a fog computing system, there is also a need to ensure dependability of the systems while at the same time guaranteeing safety of human operators as well as that of the equipment [3].

Fog computing brings benefits of cloud computing such as computation on demand through hierarchical layers of computing power between the edge devices and the cloud [4]. One such fog-based architecture for robotic systems was proposed in [5]. Here, the software components of the robotic system are distributed between the different layers of the fog architecture depending on latency and/or other functional requirements. For example, the low-level controller is usually (physically) close to the robot, while offline motion planning and predictive maintenance algorithms are run on the edge and cloud layers, respectively. While conceptually promising, there are several challenges that should be addressed before such an architecture can be implemented in practice. For example, ensuring non interference between independent applications on shared multi-processor hardware is non-trivial [6]. Resource management activities such as assigning tasks to different processors may not be straightforward due to highly variable execution times of motion planning [7].

To facilitate further discussions on fog-based software architectures for industrial robotics, in this paper, we provide a holistic overview of different technical aspects that are necessary for a practical fog-based robotic system in Section II and discuss their relevance with respect to a robotic cell based factory automation environment and its requirements, as described in Section III and Section IV. Here, we focus primarily on virtualization, resource orchestration, multi-core memory management and real-time communication along with a discussion on dependability, safety and scalability challenges concerning fog-based robotic systems. Finally, Section V concludes the paper.

II. System architecture

We consider a three layered fog computing architecture (see Fig. 1) consisting of (i) cloud layer, (ii) fog layer, and (iii) device layer. The cloud layer provides a high computing capacity, but offers limited time predictability due to varying data transmission latencies. The fog layer provides an elastic environment in the vicinity of the origin of the data. It consists of a number of interconnected physical hardware devices (fog nodes) that are capable of hosting software applications on shared node resources. The processing power of the fog layer is less than that of the cloud layer, however, the level of time predictability is improved as the infrastructure (i.e., real-time capable fog nodes, real-time networking protocols) is designed with the prior knowledge of the performance requirements. The device layer consists of resource limited devices such as sensor and actuator devices that typically pre-process data from sensors and transmit it to the fog nodes, but also smart sensors and actuators, that are capable of communicating directly with the fog nodes.

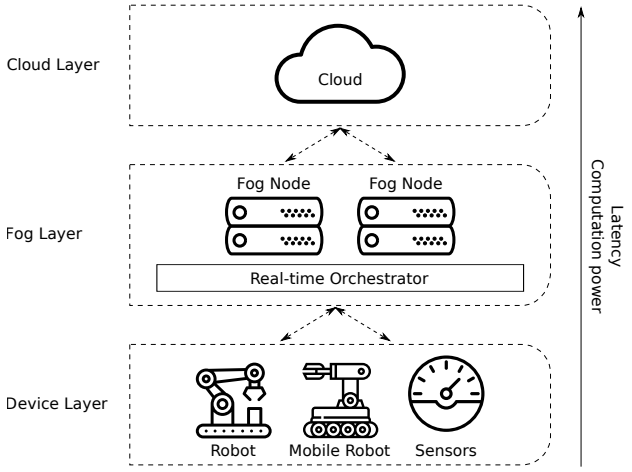

Fig. 1. Overview of a fog-based architecture.

As a fog-based industrial robotic system we denote a fog computing system enhanced with real-time capabilities, i.e., capabilities to guarantee that a computational task is finished and its result are transmitted within a predefined amount of time. Such a system should have the following properties: timeliness (the results of the computation must be finished and transmitted within a predefined time), predictability (the system must be analysable to guarantee performance of the applications), efficiency (the system should efficiently utilize available resources), scalability (the system should be able to grow in size dynamically when new cloud/fog nodes are discovered) and fault tolerance (the system should provide mechanisms to deal with unpredictable failures).

Fog Software Components and Services:

We assume a fog computing architecture to be composed of a network of fog nodes, which can be viewed as a single logical entity [8]. The network is assumed to be hybrid of wired and wireless networks to exploit the benefits of both advanced wired and wireless technologies under the practical constraints of reliability, timeliness, and security for industrial environments. We briefly discuss some of software components and services necessary for such a fog architecture below:

- System Orchestration: The system level orchestrator is responsible for ensuring that application requirements such as latency and memory are met by assignment of the applications to fog nodes. It takes into account the hardware capabilities, already running applications, task schedulability and application latency requirements. Additionally, it provides interfaces that enable seamless connection and disconnection of devices (e.g., additional fog nodes, robots, sensors and actuators) as well as interfaces for application providers for deploying applications in the fog system. Moreover, it continuously monitors the status of fog nodes in terms of availability, resource usage, and communication status and assesses the quality of service provided to the applications.
- Application Virtualization: The application virtualization component provides necessary functionality that allows to co-locate multiple independent applications on a single physical device in such a way that interference between the applications is minimized. It ensures proper allocation of resources and isolation between applications on the shared hardware building virtualized environments.
- Memory Management: Memory management component is responsible for ensuring spatial isolation among the tasks running on the same hardware. Applications have bounded memory space and cannot outbound its limits. Shared memory is allowed as long as it is explicitly declared by the applications and allowed by the operating system.
- Real-time Communication: The communication component is responsible for ensuring connectivity between the nodes and the sensor and actuator interfaces to ensure real-time data freshness, correlation and separation constraints of the applications [9]. It is also responsible for non-real-time communication and supports both wired and wireless communication.
- Scalability: Services related to scalability are responsible for adding new physical and virtual compute nodes (as well as sensing and actuating devices) to the system in a dynamic manner. Due to such scalability services, the system level orchestrator is agnostic of discovery and integration mechanisms, but is able to consider all the available computational resources for the execution of the applications.
- Dependability and Safety: The dependability services provide a set of functionality for ensuring system dependability including the safety aspects by providing means for *fault prevention, fault tolerance, fault removal* and *fault forecasting* [10]

III. Factory Automation Environment

A common factory automation setup is composed of a set of robotic cells [11]. A robotic cell consists of either a single robot or a set of robots grouped together with additional non-robotic machines to accomplish a task such as painting and welding. We categorize multi-robot cells as (i) coordinated cell, (ii) uncoordinated cell and (iii) mixed cell. In a coordinated cell, all the robots work in a synchronized manner on a single object. Here, the software components such as motion planners can be shared by all the robots. In an uncoordinated cell, the robot may work on different objects and need not be synchronized. For example, a pick and place cell with two robots operating on objects on two different conveyors need not be in sync with the other. This implies that the robots require independent software components. In a mixed cell, the robots may be in sync as well as out of sync with each other. In such a case, some of the software components are independent such as the trajectory generation component while others such as the communication component are shared. In all the cases, a supervisory controller such as a programmable

logic controller may control the workflow between the robots and other machines within the cell.

Each robot within a cell may be fitted with additional sensors and actuators. Examples of sensors are seam tracking lasers, force sensors, or vision systems, while actuators are typically end-effector tools, such as an arc torch. The sensors and actuators are physically connected to an interface device, which processes the sensor data for transmission over a real-time network. The interface device is responsible also for the processing of information from the robot controller to the actuator. It is possible that some of the sensors and actuators can be wirelessly connected to the robot controller via a wireless real-time network. In some cases, sensors and actuators are still physically attached to the interface device but the interface device itself can communicate wirelessly.

At the robot level, the runtime robot behaviour is directed through a task specification interface, where a user typically specifies the way-points that the robot should pass through, the maximum speed the robot is allowed to take, along with other attributes such as, if the robot should pass through the way-points or just within a range of the way-points. The user is also able to define logical behaviour such as to wait until a specific signal is set or a timer has expired before moving to the next way point. Finally, depending on the configuration and the user task specification, the robot software determines the trajectory of the robot using motion planning and trajectory generation algorithms [12], [13]. The information from the trajectory generation is fed to a low-level controller that runs periodically, usually, with a fixed cycle time having a typical value of 1 millisecond to control the joints of the robots [13].

To achieve this, a robot controller software, composed of diverse components, is systematically put together to provide a coherent mechanism for manipulator control supported by a real-time operating system. The main components of a typical robot controller software are motion planning and control. These are augmented by additional components for real-time and non-real-time communication. Unlike in regular real-time systems, where the execution behaviour can be modelled via different task models based on worst case execution times, modelling the execution time of motion planning tasks is complicated. The reason for this variability is twofold. One, the user of the robotic system is free to program the robot motion as desired. Two, the non-deterministic nature of the motion planning and trajectory generation algorithms. Some commonly used planning algorithms such as the Probabilistic Road Map (PRM) sample the joint space to find a collision free path [12]. In the best case, if a connection between initial point and the target point is established without collisions in the first iteration, no further computation is required. The execution time in this case can be minimal. While in the worst case, possibly in the presence of multiple obstacles, the number of samples that need to be checked for connectivity and collision can be huge, requiring larger computational time. This makes the motion and trajectory planning algorithms non-deterministic [7]. Given such a scenario, using traditional worst-case execution time (WCET) based analysis and design can result in significant wastage of resources. Additionally, if multiple robots are to be controlled using shared resources, it is difficult to guarantee that each planner will get sufficient CPU time for finding a feasible path in time (when such a path exists). The variable execution and arrival behavior limits the ability to provide response time guarantees to other real-time tasks that may be co-executing on the same hardware.

IV. Use-case Aspects

Following Section II, we further elaborate in detail on the elementary set of aspects of fog computing in the context of factory automation environments.

A. Virtualization and Real-time Aware Orchestration

1) Virtualization: The industrial use-case demands co-locating applications such as trajectory generators and communication components in a shared fog computing architecture consisting of a number of heterogeneous devices. To host such applications, the fog nodes must provide virtual environments that ensure a proper resource allocation and the fulfilment of the demands of the applications. Virtualization abstracts physical hardware from the applications running on that hardware, and thus, emulates computing environments in such a way that it appears for the applications that they are executed exclusively on a dedicated hardware. It allows to host multiple isolated applications and their software dependencies on a single physical device, and thus, reducing resource wastage. However, sharing resources may lead to (time) unpredictability, and consequently, the timing constraints may be violated.

There are two main classes of virtualization technologies: Hypervisor-based virtualization and container-based virtualization [14]. Hypervisor-based virtualization utilizes a hypervisor that distributes resources among virtual machines. This solution introduces non-negligible overheads and performance degradation mainly due to the need of full operating systems in each of the virtual machine. Thus, the amount of virtual machines on a single physical device is limited. Hypervisor-based virtualization provides stronger resource isolation [15] and minimization of interference between virtual machines. In contrast, the container-based virtualization relies on functionalities provided by the host operating system. It offers near native performance, rapid startup times and low overhead. These benefits are useful in fog computing systems as they allow to deploy a higher amount of applications in fog nodes. However, container-based virtualization provides a lower level of resource isolation and therefore lower level of time predictability of the computation. Currently, the primary objective of both the virtualization alternatives is to provide a defined amount of resources to the virtualized applications without emphasis on time-predictability.

There are few solutions addressing time predictability in hypervisor-based virtualization, e.g., RT-Xen [16] or PikeOS[1]. However, time predictability in container-based virtualization is a novel topic. As summarized in [17], real-time behavior

[1]https://www.sysgo.com/products/pikeos-hypervisor

of container-based virtualization must be supported by a predictable host operating system and real-time scheduling policies that are container-aware. The first is addressed by the application of a real-time patch that makes the Kernel fully preemptive [18] or by the use of a real-time co-kernel that runs side-by-side with a standard Kernel. Real-time aware scheduling policy for containers is addressed by utilization of hierarchical scheduling that provides temporal isolation of containers [19].

For a full adoption of real-time virtualization, we see the following challenges: (i) We have to minimize the interference between virtualized applications (e.g., co-located memory or cache-intensive applications may experience performance degradation of the physical fog nodes). (ii) There is a lack of real-time communication mechanisms between virtualized applications (that may be executed on different devices) and enabling the communication in a time-predictable, secure, and safe manner. (iii) The possibility of supporting a mixture of both hypervisor-based virtualization (to satisfy hard real-time requirements) and container-based virtualization (to satisfy less stringent requirements) in a single fog computing system should be explored. (iv) Deal with unpredictability of communication between virtualized applications in a fog computing architecture due to network performance.

2) Real-Time Aware Orchestration: The role of the orchestrator in the fog-based industrial computing systems is to maintain the deployment of virtualized applications (either in containers or virtual machines) in the shared fog and cloud computing environment in such a way that the resource and timing requirements are fulfilled. The main phases that an orchestrator should cover are: resource selection, service deployment, service monitoring and resource control [20]. Although, there has been extensive research on orchestration, taking into account various resources and optimization goals, and there are several mature orchestrator systems available[2], none addresses real-time related requirements. Therefore, we envision the following real-time enhancements of the orchestrator that can serve in fog-based industrial computing systems. It should enhance the orchestrators in: *a)* resource selection, *b)* real-time deployment, *c)* real-time aware service monitoring and *d)* real-time resource control.

a) Resource Selection: The orchestrator must be aware of timing requirements of applications and real-time capabilities provided by fog nodes. The orchestrator must perform schedulability tests that ensure that the virtualized applications will be granted enough CPU time for performing time-critical actions. Additionally, the orchestrator should be aware of interference between virtualized applications and try to minimize such impacts during the resource selection phase.

b) Real-time Deployment: The orchestrator should provide a bounded time for deployment of virtualized applications. It should take into account transmission times of the applications from the repository to the fog node and the startup time of the application. This enhancement is important for safety and dependability aspects, e.g., during the redeployment of a failed application.

c) Real-time Aware Service Monitoring: Due to imperfections of underlying operating systems (e.g., Linux) that may not provide accurate temporal isolation for virtualized environments, the orchestrator must monitor the quality of service delivered by the virtualized applications (e.g., deadline misses or lateness[3]). The orchestrator should use this information while resource selection phase.

d) Real-time Aware Resource Control: Based on real-time related metrics obtained in the previous point, the orchestrator should perform migration of virtualized applications in order to improve their real-time behavior. This can be a case of a memory or cache-intensive application that may experience performance degradation when it is co-located with another memory or cache-intensive application on a single node.

B. Multi-core Platforms and Memory Management

Computational power is one aspect that needs significant improvements to support the demands of new robotic systems applications. One way to increase the computational power is to utilize multi-core platforms as fog nodes. However, general Commercial Off-The-Shelf (COTS) Multi-Core Processors (MCP) share hardware resources like cache and main memory. Sharing such resources is one of the primary sources of a task's Worst Case Execution Time (WCET) unpredictability [21]. In an MCP, the task execution time not only depends on the task itself, but it is also significantly influenced by applications running on the other cores. Nowotsch *et al.* [6] showed that the latency of a single memory store request can increase up to 25.82 times when the number of active cores increases from 1 to 8. Since virtualization allows co-execution of independent applications such as the control tasks of different robots on the same hardware, bounding the WCET is necessary for the use of schedulability analysis and admission tests by the orchestrator. Such analysis allows the orchestrator to optimally allocate the resources for the applications.

Multiple solutions have been proposed to improve and also to guarantee the tasks' WCET on an MCP in the context of real-time systems. Yun *et al.* [22] propose a memory bandwidth management system called MemGuard. It is implemented in the operating system layer and divides the available memory bandwidth in two components, i.e., guaranteed and best effort. In order to guarantee minimal memory bandwidth to each core, and achieve performance isolation, the MemGuard is based on the guaranteed component. The best effort component takes the difference between the total available memory bandwidth of the system and the bandwidth reserved for the guaranteed part. The hardware platform provides a Performance Monitor Counter (PMC) unit that allows counting hardware-related events in the core. The PMC, in each core, is programmed to account for the memory access usage, and

[2]e.g., Kubernetes (https://kubernetes.io/), Docker Swarm (https://docs.docker.com/engine/swarm/), or OpenStack(https://www.openstack.org/

[3]The delay of a task completion with respect to its deadline.

raises an interruption when it reaches the limit keeping the specified memory bandwidth. To improve memory bandwidth utilization, the authors propose a reclaiming mechanism leveraging usage prediction on each core. Due to the reclaiming algorithm being based on prediction, the proposed system is intended to support mainly soft real-time applications.

Agrawal *et al.* [23] extend [22] and propose a dynamic memory bandwidth isolation using a number of memory bandwidth levels. The method is a runtime mechanism integrated with a global scheduler that uses two servers running in each core. One to regulate the execution time, and the other to regulate the total number of memory accesses. An off-line schedule table is computed to find the server budget for each server on its respective core, and the global scheduler sets the budget to the corresponding server.

The presented solutions based on time and memory bandwidth management are well suited for time-triggered applications in a context where we know the number of active cores at the same time, and the maximal contentions introduced by these cores. It becomes unrealistic for a industrial robotic system where the number of applications and their requirements change dynamically and during runtime as, for instance, regarding the nature of the robot motion planning and trajectory generation algorithms cited in Section III. Therefore, to achieve real-time guarantees in a system running on a MCP, we have (i) to ensure the temporal isolation of tasks and containers taking the platform resource contentions into account, and (ii) to design a resource sharing mechanisms for managing access to shared resources, such as bus and memory controller, taking dynamic applications scenarios into account.

However, it is also unrealistic to assume that the maximum memory bandwidth necessary for each legacy critical task is known. The worst-case number of memory accesses that an application can issue depends on many different factors, for instance, hardware platform, system configuration, and operating system. To address the lack of this data, an alternative solution is to bind the containers that run critical tasks to a specific core and monitor their execution progress in predefined checkpoints. In case execution is late, the other non-critical containers can be paused to reduce the inter-core interference preventing the critical one from missing a deadline.

COTS multi-core processors are designed primarily for the average-case performance and that is not enough to meet the real-time requirements of robotic applications. Therefore, to address the loss of predictability in this kind of hardware platforms, we need to apply new mechanisms and know better the application resources needs.

C. Timely and Reliable Communication

In the system architecture used in this paper, the fog nodes communicate with a number sensors, actuators and other devices in different layers, using both wired and wireless connections to ensure the smooth operation of the fog-based robotic system. However, when the probability of losing a packet goes up, e.g., due to the noise in industrial environments combined with the Doppler effect, multi-path fading, and dynamic wireless channel [24], this can lead to an increase of end-to-end latency. Therefore, to enable fog-based industrial robotic systems, subject to real-time requirements, the communication protocols should be designed to fulfil strict timeliness and reliability requirements, i.e., an upper bound of end-to-end latency of 1 millisecond along with a probability that a packet does not reach its destination before the deadline to not exceed 10^{-7} [25]. While wired networks can offer such deterministic communications, wireless ones should guarantee deterministic reliable communications at the same level.

In this context, Medium Access Control (MAC) protocols and relaying strategies are central in achieving the desired requirements. Rajandekar et al. [26] concludes that hybrid MAC protocols can meet the stringent requirements on reliability and timeliness. Moreover, there is a massive connection at the fog layers, as presented in Section IV-D, thus, the proposed MAC protocols must support a large number of simultaneous connections. Li et al. [27] proposes a hybrid Time Division Multiple Access (TDMA) Non Orthogonal Multiple Access (NOMA) scheme for cellular-enabled machine-to-machine communications. With NOMA, multiple nodes can be served simultaneously utilizing the same time-frequency resources but different power levels. A proposed time-sharing scheme is introduced to deal with the massive deployment of devices, while the total transmission time and energy efficiencies are obtained. This solution is suitable for fog networks where a NOMA transceiver may be deployed at the fog nodes [28]. Another approach, Hoang et al. [29] proposes a relaying sequence that considers all cases that can happen in each time slot. Hence, the probability of an error is an exact value compared to an upper bound value as is the case in other solutions related to multi-hop communication. Moreover, the authors introduce a method of group based relaying on a hybrid TDMA-CSMA (Carrier-Sense Multiple Access) protocol to address the drawback in relaying sequencing where a relayer keeps silent if it does not have any correct copy of the packet [30]. Therefore, the relaying strategies combined with packet aggregation are practical techniques that can be implemented on COTS devices as well as fog nodes. However, these techniques, that are based on a hybrid protocol with NOMA, need to be exploited further to reduce the end-to-end latency coming with a massive connection at the fog layer.

To obtain timely and reliable communication, we observe several challenges including: (i) the placement of fog nodes must minimize the probability of error and end-to-end latency, (ii) finding the number of fog nodes that meet all constraints on timeliness and reliability should be found, for each specific application, (iii) hybrid protocols with NOMA should be studied to deal with the massive connection, and (iv) in industrial environments, there are strict constraints, i.e., a limited number of relay nodes and re-transmissions - therefore, specific relaying strategies should be defined for each application.

D. Scalability

Another aspect of the proposed use case is scalability [31]. In an industrial setting (as discussed in Section III), the number of participating fog nodes can become very large depending on the size of the factory because the network involves not only the fog nodes but also sensors, actuators and the interface devices. Furthermore, apart from interconnecting the compute nodes of one factory, there is also benefit from interconnecting various factories with each other. This may increase the flexibility and efficiency of the production lines due to considering a larger pool of available resources. This becomes evident, for instance, when the production of an item in a specific factory is reduced or halted due to potential faults in the hierarchical or vertical communication of the nodes (see Section IV-E). In such cases, other factories can change their production plans in order to compensate for the faults. This can be achieved by coordinating the function of multiple factories through fog and cloud nodes.

Along with the potential benefits of interconnecting factories using fog and cloud nodes, and the advantages that fog computing will bring to industrial environments, there are also related concerns. In order to create such a scalable industrial setting, the underlying communication mechanisms need to be able to scale without generating a significant amount of overhead which may hinder the operation of the applications [31], and compromise the functionality of nodes which execute tasks with strict deadlines (see Section IV-B). Most current approaches that can be used for fog computing aim at providing the necessary communication mechanisms to allow computing close to the edge of the network [32]. However, they do not consider scalability metrics (e.g., generated overhead) which show that scaling a fog computing system to a large degree does not compromise the performance of the applications. Additionally, the runtime management and orchestration of virtualized resources need to trade off the reconfiguration overhead with an optimal allocation of the available resources [33]. This is also impacted by safety and fault-tolerance mechanisms, that are often based on redundancy (see Section IV-E).

In an industrial fog computing system, when adding new compute nodes (e.g., new controllers, fog, and cloud nodes) these nodes need to be discovered and integrated in the system [34]. This means that the existing nodes need to store the new nodes' information (e.g., IP addresses, amount of computational resources, etc.) so that they can communicate (e.g., using widely-used communication protocols [35]). This creates the problem of determining which nodes a new node should connect to [36]. A simple solution to this problem would be that each node maintains a global view of the system, i.e., stores the information of all the other nodes. However, this means that in a system with a growing number of nodes (such as an industrial setting in which new nodes may be added dynamically at any time), each node needs a growing amount of storage resources, to store the necessary information. This is unrealistic because some of the participating compute nodes may have limited computational resources. For this reason, scalability still poses a challenge in the proposed use case.

E. Dependability and Safety

Dependability: While fog-based robotic systems address some of the limitation of existing architectures [5], they introduce new challenges for ensuring dependability attributes, for instance, reliability and availability, which are easier to address in the existing architectures due to their dedicated resource usage and less complexity. The fog-based software architecture demands frequent data exchange between the different layers of the fog platforms as well as within the different nodes of the fog layer to accomplish a functionality such as trajectory generation and control. This puts more focus on ensuring reliability and availability of the fog platforms. As a result, reliability and availability of the fog platforms is critical for realising robotic applications on the fog platforms.

Different attributes of these services are challenged by dependability threats known as errors, faults and failures [10], which in turn might disrupt the entire functionality of the system. Threats related to computational resources depict the occurrence of faults in main components in the fog layer, i.e., fog nodes and the system orchestrator each providing a set of functionalities for robot motion. Dependability threats related to the orchestrator may lead to either a performance failure or waste of resources. A failure in data storage might cause data loss. This can result in loss of user-specified tasks or the configuration settings mapping different sensors to the user task specifications and data variables. Additionally, data stored for future analysis in the cloud will not be accessible. Threats related to communication are any faults that might disrupt or prevent the connectivity between the different nodes as well as the different layers in the architecture. Loss of communication between the nodes and the edge devices can result in stoppage of robots as the new trajectory parameters are not available for the low level controllers. Such failures can have a cascading effect within a robotic cell if the failure is local to a robotic cell, especially that of a coordinated cell. It is therefore critical to preserve a tight end-to-end communication, while providing suitable fault tolerance mechanisms.

Dependability approaches for fog computing in the literature are mainly proposed to address dependability requirements using fault management solutions and redundancy techniques [37]. Fault management solutions proposed are mainly considering threat detection tools like monitoring [38] for fault prevention and fault removal, or failure recovery solutions like reconfiguration up on failure [39]. Redundancy techniques proposed in the literature focus on different addressing different dependability requirements of the fog platform. For instance, improving reliability [40], and availability [41], of the system by implementing redundant components such as a redundant node [40], in a network, redundant communication channels [42] and task offloading [43] or application migration [44].

The proposed dependability solutions are mainly tied to redundancy methods and replication techniques, for instance,

TABLE I
OVERVIEW OF THE ASPECTS AND THEIR INTER-RELATION AND CHALLENGES.

Aspect/Relation	Virtualization & Orchestration	Multi-Core Platform	Communication	Safety & Dependability	Scalability
Virtualization & Orchestration	–	Platform abstraction and efficient resource management.	Resource allocation for achieving tight end-to-end communication.	Provide predictability and fault tolerance.	Complexity.
Multi-Core Platform	Provide real-time guarantees and predictability.	–	Virtualization & Orchestration isolates the platform.	Intra-core isolation and predictability.	Challenges wrt. hierarchical architectures & legacy.
Communication	Heterogeneity, combination of wired and wireless networks.	Virtualization & Orchestration isolates the platform.	–	Trade-off between reliability and safety.	Large numbers along with big data communication.
Safety & Dependability	Heterogeneous safety levels.	Certification.	Tight end-to-end communication and fault tolerance.	–	Redundancy overhead.
Scalability	Runtime reconfiguration.	Hardware architecture.	Congestion.	Increased complexity.	–

using passive/ active replicas of the system components. However, using replicas for each component will result in overhead of cost and consumption of resources [45], and it can impacts the scalability of the overall system.

Safety: Safety in robotics involves multiple domains such as the design of the manipulator arm and layout of the cell. We focus on safety from the software perspective. We define safety in the present use-case as the property of the system which guarantees the timely and correct execution of safety-critical tasks (with hard real-time deadlines) under *all* operational conditions [46]. This encompasses scenarios wherein the system reverts to a safe state in the event of a safety violation or hazard. Catastrophic consequences (such as loss of life, danger to the surrounding environment) can ensue in case of a failed execution of these safety-critical tasks occur [47]. For example, in a factory automation environment, safety implies that the control signals issued to the robot arm do not lead to motion primitives that can cause the arm to move in a non-deterministic manner. To ensure that such faults are well handled, it is necessary that timing analysis, schedulability tests and the network schedule are designed considering the potential threats to safety. This poses one of the main challenges for the verification and validation methods for multi-core platforms, and more in general for fog architectures.

For a thorough analysis of the fog computing system we need to ensure such threats are handled at the design phase itself, even in presence of heterogeneous safety levels. Additionally, as in all safety systems, there needs to be no single point of failure. Along with these challenges, it is necessary to investigate if real-time applications running in the fog provide the same level of safety guarantees that existing safety-certified robotic systems provide.

V. CONCLUSION

Fog-computing brings cloud-like capabilities to low latency applications but the practical implementation of a fog architecture for real-time applications such as robot control is non-trivial. To move a step forward in this direction, we need a holistic approach that considers different technical aspects independently but also in conjunction with each other. We summarize the relationship between the different aspects in Table IV-D. From an infrastructure point of view, we consider the use of COTS-based multi-core processors as fog nodes for providing the computational capabilities.

To make effective use of these resources, we need virtualisation techniques such as hypervisors and containers with real-time capabilities to support the timing requirements of robotic cells and robot motion. To provide temporal isolation, hypervisors and containers need memory management techniques to limit the interference of shared caches and buses within the multi-core architectures. By providing bounds on the interference, we can enable the orchestrator with the capabilities to use real-time schedulability analysis and appropriate scheduling algorithms to allocate applications to fog nodes, to ensure timing predictability. Since the fog platform is a distributed system involving fog nodes and edge devices such as low-level controller and sensors, we need real-time communication mechanisms. Such communication can be wired or wireless. The constraints imposed by communication technologies further guide the scheduling and allocation of resources by the orchestrator. For robotic applications, dependability and safety are important attributes to prevent any damage to the equipment and more importantly, to safeguard the health of the operators working in close proximity to such robots. To this end, we need solutions that consider the requirements of the applications as well as the new challenges imposed by the fog platforms. In this paper, we briefly discussed a robotic cell environment to highlight the usefulness of fog-based solutions and discussed key aspects such as resource orchestration and network scalability, virtualization and memory management techniques supported by real-time communication paradigms. Further we discussed the dependability and safety issues that need to be considered when moving towards the fog-based architectures for robotic applications.

REFERENCES

[1] M. Hägele *et al.*, "Industrial robotics," in *Springer Handbook of Robotics*, 2016.

[2] V. Villani *et al.*, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, 2018.

[3] J. A. Marvel and R. Norcross, "Implementing speed and separation monitoring in collaborative robot workcells," *Robot. and Computer-Integrated Manuf.*, 2017.

[4] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Workshop on Mobile Cloud Comp.*, 2012.

[5] S. M. Salman *et al.*, "Fogification of industrial robotic systems: Research challenges," in *Proceedings of the Workshop on Fog Computing and the IoT*, 2019.

[6] J. Nowotsch *et al.*, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *Euromicro Conf. on Real-Time Systems (ECRTS)*, 2014.

[7] M. Alcon *et al.*, "Timing of autonomous driving software: Problem analysis and prospects for future solutions," in *IEEE Real-Time and Embedded Technol. and Appl. Symp.*, 2020.

[8] E. M. Tordera *et al.*, "What is a fog node a tutorial on current concepts towards a common definition," *arXiv preprint arXiv:1611.09193*, 2016.

[9] R. Gerber *et al.*, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Trans. on Softw. Eng.*, 1995.

[10] A. Avizienis *et al.*, "Basic concepts and taxonomy of dependable and secure comput." *IEEE Trans. on Dep. and Secure Comp.*, 2004.

[11] J. Zhang and X. Fang, "Challenges and key technologies in robotic cell layout design and optimization," *J. Mech. Eng. Science*, 2017.

[12] S. M. LaValle, *Planning Algorithms*, 2006.

[13] T. Kröger, *On-Line Trajectory Generation in Robotic Systems*, 2010.

[14] R. Morabito *et al.*, "Hypervisors vs. lightweight virtualization: A performance comparison," in *IEEE Int. Conf. on Cloud Eng.*, 2015.

[15] M. G. Xavier *et al.*, "A performance isolation analysis of disk-intensive workloads on container-based clouds," in *Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing*, 2015.

[16] S. Xi *et al.*, "RT-Xen: Towards real-time hypervisor scheduling in Xen," in *ACM Int. Conf. Embedded Software*, 2011.

[17] V. Struhár *et al.*, "Real-time containers: A survey," in *Workshop on Fog Comput. and IoT*, 2020.

[18] A. Moga *et al.*, "Os-level virtualization for industrial automation systems: Are we there yet?" in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016.

[19] L. Abeni *et al.*, "Container-based real-time scheduling in the Linux kernel," *ACM SIGBED Review*, 2019.

[20] R. Ranjan *et al.*, "Cloud resource orchestration programming: overview, issues, and directions," *IEEE Internet Comput.*, 2015.

[21] S. Schliecker *et al.*, "Bounding the shared resource load for the performance analysis of multiprocessor systems," in *Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2010.

[22] H. Yun *et al.*, "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Trans. on Comp.*, 2016.

[23] A. Agrawal *et al.*, "Contention-Aware Dynamic Memory Bandwidth Isolation with Predictability in COTS Multicores: An Avionics Case Study," in *Euromicro Conf. on Real-Time Syst.*, 2017.

[24] A. Willig *et al.*, "Wireless technology in industrial networks," *Proceedings of the IEEE*, 2005.

[25] R. Candell and M. Kashef, "Industrial wireless: Problem space, success considerations, technologies, and future direction," in *Resilience Week*, 2017.

[26] A. Rajandekar and B. Sikdar, "A survey of MAC layer issues and protocols for machine-to-machine communications," *IEEE Internet of Things J.*, 2015.

[27] Z. Li and J. Gui, "Energy-efficient resource allocation with hybrid tdma–noma for cellular-enabled machine-to-machine communications," *IEEE Access*, 2019.

[28] H. Tezuka *et al.*, "A UL-NOMA system providing low E2E latency," in *IEEE VTS Asia Pacific Wireless Commun. Symp.*, 2019.

[29] L.-N. Hoang, "Relaying for timely and reliable applications in wireless networks," Ph.D. dissertation, Halmstad University, 2017.

[30] L. Hoang *et al.*, "Relay grouping to guarantee timeliness and reliability in wireless networks," *IEEE Commun. Lett.*, 2019.

[31] V. Karagiannis *et al.*, "Enabling fog computing using self-organizing compute nodes," in *IEEE Int. Conf. Fog and Edge Comput.*, 2019.

[32] V. Karagiannis and A. Papageorgiou, "Network-integrated edge computing orchestrator for application placement," in *Int. Conf. Netw. and Service Manage.*, 2017.

[33] W. Tärneberg *et al.*, "Distributed approach to the holistic resource management of a mobile cloud network," in *International Conference on Fog and Edge Computing (ICFEC)*, 2017.

[34] V. Karagiannis *et al.*, "Addressing the node discovery problem in fog computing," in *Workshop Fog Comput. and IoT*, 2020.

[35] ——, "A survey on application layer protocols for the internet of things," *Trans. on IoT and Cloud comput.*, 2015.

[36] V. Karagiannis, "Compute node communication in the fog: Survey and research challenges," in *Workshop on Fog Comput. and IoT*, 2019.

[37] Z. Bakhshi and G. Rodriguez-Navas, "A preliminary roadmap for dependability research in fog computing," *ACM SIGBED Review*, 2020.

[38] X. Yuan *et al.*, "Cyber-physical systems for temporary structure monitoring," *Automation in Construction*, 2016.

[39] Y. Xiao *et al.*, "A novel task allocation for maximizing reliability considering fault-tolerant in VANET real time systems," in *IEEE Int. Symp. on Personal, Indoor, and Mobile Radio Communications*, 2017.

[40] A. Aral and I. Brandic, "Quality of service channelling for latency sensitive edge applications," in *IEEE Int. Conf. Edge Computing*, 2017.

[41] X. Chen *et al.*, "A fault-tolerant data acquisition scheme with mds and dynamic clustering in energy internet," in *IEEE Int. Conf. Energy Internet*, 2018.

[42] K. E. Benson *et al.*, "Ride: A resilient IoT data exchange middleware leveraging SDN and edge cloud resources," in *IEEE/ACM Third Int. Conf. Internet-of-Things Design and Implementation*, 2018.

[43] M. T. Saqib and M. A. Hamid, "FogR: A highly reliable and intelligent computation offloading on the internet of things," in *IEEE Region 10 Conf. (TENCON)*, 2016.

[44] O. Osanaiye *et al.*, "From cloud to fog computing: A review and a conceptual live vm migration framework," *IEEE Access*, 2017.

[45] Z. Bakhshi *et al.*, "Dependable fog computing: A systematic literature review," in *Euromicro Conf. Software Eng. and Adv. Appl.*, 2019.

[46] R. Dobrin *et al.*, "On fault-tolerant scheduling of time sensitive networks," in *Int. Workshop on Security and Dependability of Critical Embedded Real-Time Syst.*, 2019.

[47] N. Desai and S. Punnekkat, "Safety-oriented flexible design of autonomous mobile robot systems," in *Int. Symp. on Syst. Eng.*, 2019.