A Time-Predictable Fog-Integrated Cloud Framework:

One Step Forward in the Deployment of a Smart Factory

Hamid Reza Faragardi*, Saeid Dehnavi*†, Mehdi Kargahi†, Alessandro V. Papadopoulos*, Thomas Nolte*
*MRTC / Mälardalen University, Västerås, Sweden
{hamid.faragardi,alessandro.papadopoulos, thomas.nolte}@mdh.se
†School of ECE, College of Eng., University of Tehran, Iran
{sdehnavi,kargahi}@ut.ac.ir

***Abstract*—This paper highlights cloud computing as one of the principal building blocks of a smart factory, providing a huge data storage space and a highly scalable computational capacity. The cloud computing system used in a smart factory should be time-predictable to be able to satisfy hard real-time requirements of various applications existing in manufacturing systems. Interleaving an intermediate computing layer –called *fog*– between the factory and the cloud data center is a promising solution to deal with latency requirements of hard real-time applications. In this paper, a time-predictable cloud framework is proposed which is able to satisfy end-to-end latency requirements in a smart factory. To propose such an *industrial cloud framework*, we not only use existing real-time technologies such as Industrial Ethernet and the Real-time XEN hypervisor, but we also discuss unaddressed challenges. Among the unaddressed challenges, the partitioning of a given workload between the fog and the cloud is targeted. Addressing the partitioning problem not only provides a resource provisioning mechanism, but it also gives us a prominent design decision specifying how much computing resource is required to develop the fog platform, and how large should the minimum communication bandwidth be between the fog and the cloud data center.**

***Keywords*-Smart factory; cloud computing; fog computing; edge computing; resource allocation, real-time applications.**

I. Introduction

In order to make modern production cost-efficient, future production lines need to be smarter and more flexible [1], that is the principal notion of Industry 4.0 [2]. Industry 4.0 creates what has been called a *Smart Factory*. To achieve this goal, all the manufacturing processes are supposed to be configurable, and connected to the Internet. The idea can be fulfilled through connecting and integrating traditional industries, by providing communication between producers and consumers. This idea in near future will revolutionize the whole industrial panorama, as pointed out by the Fraunhofer Institute [3].

The integration of three main elements, including Cyber-Physical Systems (CPS), the Internet of Things (IoT) and cloud computing, builds the foundation of a smart factory. In a smart factory, we encounter with an exponential increase of data size and computational complexity in comparison to traditional manufacturing factories. This is due to (i) participation of a higher number of nodes, each of which continuously generates a stream of data that needs to be stored and processed, and (ii) machine to machine interactions which relies on multiple modern technologies such as big data analysis [4] and cognitive computing [5] which both demand a huge amount of data storage and processing. A promising solution to address the huge data size and extensive computations is cloud computing. If the required computing resources are supplied only by local resources within the factory, then both the cost of purchasing and the maintenance cost dramatically increase the Total Cost of Ownership (TCO). A considerable increase of TCO hinders the development of a smart factory in terms of finances. Hence, cloud computing is adopted as one of the main components of a smart factory to provide highly scalable computing and storage capacity.

Most of the manufacturing applications contain multiple hard real-time requirements. However, today's cloud providers neither provide a guarantee for hard real-time applications, nor provide a possibility for users to specify the deadline of their applications. Nevertheless, recently multiple solid solutions are proposed to provide a real-time cloud data center [6], [7]. They have made a foundation for developing a time-predictable cloud framework by this paper.

Even if we provide a time predictable cloud data center, which is able to guarantee the real-time requirements of cloud services, a sub set of real-time applications with tight latency requirements (we call such applications, *hard applications*) can still not run in the cloud.

The reason is inherent in the bandwidth limitation of the communication lines between the cloud data center and the factory. Here, a considerable time is spent over the network for exchanging the data, introducing a noticeable delay in response time of the applications running in the cloud. Therefore, we need to extend the cloud framework to cope with such hard applications having tight latency requirements. Along with real-time requirements, there are other principal challenges such as availability and security in outsourcing workload (data and computations) from local servers to a cloud data center in a smart factory.

An effective solution to deal with all the above mentioned challenges (hard real-time applications, security, availability) is to retain a portion of a given workload on local resources, a notion which is referred to as *Fog Computing*– while the rest of the workload is outsourced to a cloud data center. This vertical extension of the cloud scheme is called fog, because a fog is a cloud that is closer to the ground. Similarly, in a smart factory, a fog platform which is located in the factory and connected with a local network is physically closer to the factory in comparison to a cloud data center which is

978-1-5386-1475-4/18/$31.00

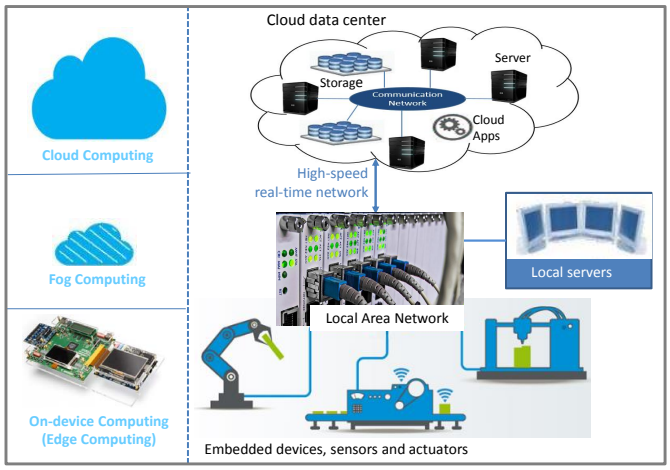

Fig. 1. Flowchart representation of our proposed solution framework.

located externally. A fog is constructed by a set of fog nodes where a fog node could be either the remaining capacity of a network device such as routers, mobile stations, gateways, and Software Defined Network (SDN) controllers or it could be a local server. Although interleaving fog into the industrial cloud framework makes benefits for security, availability and timeliness of the system, this paper concentrates only on the real-time dimension of the problem. Figure 1 shows an example of a smart factory in which three computing layers including edge, fog and cloud computing are marked.

A. Contributions. In this paper, we first propose a time-predictable cloud architecture by integrating fog resources. We also introduce a resource provisioning mechanism to partition a given workload among cloud computing resources, fog resources and on-device resources. The goal of resource provisioning mechanism is to reduce the deployment cost of a smart factory by

1) using the maximum capacity of available computing resources in a smart factory,
2) pushing the remaining workload that does not fit in the local available resources to the cloud, and
3) only for the remaining portion of the workload that neither fits in the local resources, nor can be executed on the cloud (because of hard timing requirements), a minimum additional computing resource, called local servers, is integrated to the system.

In other words, we intend to minimize the number of required local servers, while maximizing the utilization of available local resources without sacrificing real-time requirements. This strategy decreases the TCO as a result of (i) a lower cost of the deployment of the system, (ii) a lower maintenance cost as a result of pushing the feasible part of the workload on the cloud.

To the best of our knowledge, the partitioning of a real-time workload among the different computing layers in a smart factory including edge, fog and cloud computing has not been addressed so far. Moreover, the minimum size of the fog required to satisfy the strict timing requirements of industrial applications has not been sufficiently discussed in the literature.

B. Organization of the paper. The rest of this paper is organized as follows: In Section II a brief survey of related work is presented. A time predictable cloud framework is proposed in Section III where a Fog platform is integrated into the cloud framework to deal with hard applications in a smart factory. The problem is described in detail and assumptions are defined in Section IV. Section V introduces a resource provisioning mechanism for partitioning of a given workload among the cloud data center and the Fog platform in the proposed cloud framework. In Section V the performance of the proposed mechanism is evaluated and the impact of different system configurations on the vertical placement decisions is investigated. Finally, the concluding remarks and future work are discussed in Section VI.

II. Related Work

Virtualization techniques have been widely studied in the real-time scheduling community. Early works have evolved around virtualizing the computational capacity of single-processor hardware. Such a virtualized hardware is refereed as a Virtual Processor (VP). For instance, the periodic resource model uses period and budget [8] for characterizing the VPs. Several authors have proposed different virtualization models targeting multiprocessors. Multiprocessor models need to specify the maximum parallelism level in their interface. Shin *et al.* proposed the Multiprocessor Periodic Resource (MPR) model [9]. Lipari and Bini suggested the Bounded-Delay Multipartition (BDM) model [10]. Leontyev and Anderson [11] proposed a model that only specifies bandwidth w in the component interface. The Generalized MPR (GMPR) model [12] reduces pessimism of the MPR model. There is often a trade-off between simplicity and accuracy in virtualization models. In other words, simple interfaces tend to be pessimistic and introduce more resource loss than the more detailed yet complex models. We opted the MPR model because of its simplicity. Scheduling of VMs compliant with the MPR model is implemented in the RT-Xen hypervisor [13].

From a placement perspective, a wide range of studies have been carried out to place a set of Virtual Machines (VMs) into the physical servers to minimize operation cost of a cloud data center [14]. The majority of these works consider real-time aspects of VMs while the real-time requirement is either refereed as the SLA [15] or explicitly mentioned as deadlines [16].

Recently, [17], [18] considered the placement of VMs to servers in a cloud data center to minimize energy consumption by using a consolidation approach while live migration was also taken into account. In addition, in some of these works, e.g. [19], energy consumption of the network equipment is also taken into consideration.

Although extensive studies have been carried out in the context of VM placement onto the cloud data centers, most often the structure of their real-time VMs are different compared to what is considered in this paper. The VMs in the mentioned works consist of a set of ordinary real-time tasks where there is only one instance for each task whereas in

this paper we consider periodic real-time tasks where a set of identical instances of a task is released periodically. It leads us to an additional complexity.

There is a recently proposed approach called RT-OpenStack [7] considering a similar task model for the cloud applications as what we do in this paper. They consider global Earliest Deadline First (gEDF) as an guest scheduler within each VM and also use compositional schedulability analysis to specify the demand of VMs in terms of the number of processors and required budget of VMs. They use RT-XEN as the VMM where gEDF is applied to allocate VCPUs to the PCPUs of the hosts.

A comprehensive survey on fog computing is proposed by [20]. They map the existing works to the taxonomy in order to identify current research gaps in the context of fog computing. The fog computing paradigm and its role on the IoT are introduced by [21]. They also delineate the characteristics of fog computing, and the platforms that support fog services. In [22] a fog computing supported software-defined embedded system is presented, where computations can be performed on either embedded devices or on a computation server. The task scheduling and resource management are conducted such that task completion time is minimized for promoting the user experience. In [23] an overview of the core issues, challenges, and future research directions in fog-enabled orchestration for IoT services are provided. Additionally, it presents early experiences of an orchestration scenario, demonstrating the feasibility and initial results of using a distributed genetic algorithm in this context. However, they mainly focus on the communication parts rather than computations, hence their task model is different compared to what we consider in this paper.

In [24], they propose an online algorithm for computation offloading in Mobile Edge Computing (MEC) systems with energy harvesting devices. Liu et al. [25] present a Markov decision process approach to handle computation offloading in MEC, where the computation tasks are scheduled based on the queuing state of the task buffer, the execution state of the local processing unit, and the state of the transmission unit.

In [26], Villari et al. propose the osmotic computing paradigm. The aim of osmotic computing is to obtain a balanced deployment of microservices, meeting low-level constraints and high-level needs, such as load balancing, reliability, and availability. Our paper extends this framework targeting real-time capabilities of the infrastructure, including constraints on the computational and communication delays, and real-time deadlines.

III. A TIME-PREDICTABLE CLOUD FRAMEWORK

In this section a time-predictable cloud framework is proposed; suitable to be applied in an industrial setting. Additionally, unaddressed challenges are discussed.

A time-predictable cloud framework is defined as a cloud-based system in which the time duration of completion of an operation from the start to the end time (i.e., end-to-end latency) is not longer than a specific time interval. Indeed, an end-to-end latency reflects a time interval beginning once a sensor of a CPS in a factory (e.g., a robot) sends data, then processing on the data is performed, until the result of the processing returns to the actuator of the same (or different) CPS. In the first step, local computing/storage resources within the embedded device – called edge node – is used to process the data. However, since the computing/storage capacity on the edge node is limited, the data may be sent to fog nodes through the local network. Then, the processing on the given data is performed either in the fog nodes or the data is sent through the dedicated communication lines to a cloud data center where the processing is performed. Eventually, wherever the data is processed (i.e., edge, fog or cloud), the output of the processing should be returned to the actuator(s) on the edge layer.

To propose a time-predictable cloud framework in a smart factory, the following components should be considered:

1) A real-time local communication network in a smart factory inter-connecting CPSs and the fog nodes,
2) A time-predictable resource scheduling mechanism within each fog node and edge node,
3) A sufficient amount of fog resources in terms of both the number of nodes and the processing capacity of them,
4) A resource scheduling mechanism to distribute the workload among the fog nodes,
5) A high-speed real-time communication network connecting the smart factory to a cloud data center,
6) A proper cloud interface prepared by the cloud provider to reflect the demand of real-time applications,
7) A vertical resource provisioning mechanism specifying the allocation of a given workload among different computing resources, i.e., on device resources, fog resources and the cloud.

Four of these seven components have been extensively discussed in the literature. However, item 3, 4 and 7 have not been adequately investigated yet. Let us start with the four addressed components and then we will have a look at the remaining items.

For the first item, communication between sensors, actuators, and the control center is done through an industrial network which is a real-time local network. Fortunately, in most modern manufacturing factories, such a real-time network has already been deployed by using reliable, secure and time-predictable network mechanisms and protocols such as the HART protocol [27] and Industrial Ethernet. Industrial Ethernet protocols like PROFINET and EtherCAT [28] modify standard Ethernet in a way to ensure that manufacturing data is not only sent and received correctly, but also sent and received on-time. Industrial Ethernet is also able to handle factory noise, factory process needs, and harsher environments [29]. In a smart factory, according to the fundamental concept of IoT, a sub set of objects (specially sensors) could connect to the system over a wireless network. In other words, wireless and wired stations are integrated into one single network, called a hybrid network. For the wireless part of an industrial local network, there also exist reliable, secure and time predictable protocols and mechanisms. WirelessHart [30] and the IETF 6TiSCH Working Group (WG) [31] are well-known examples

of such industrial wireless communication mechanisms.

For the second item, a similar approach as proposed by [7] is adopted as a resource provisioning mechanism to yield a time-predictable Fog platform. They start with assigning a set of real-time applications to VM, referred as real-time VM. Their approach support a resource interface that allows a real-time VM to specify the amount and temporal granularity of CPU resource allocation needed to meet the real-time performance requirements of its applications. The amount and temporal granularity of the required CPU resource is specified according to compositional schedulability analysis [32]. Afterward, a real-time-aware VM-to-Host mapping is presented to maintain the schedulability of real-time VMs without overloading the servers. The mapping of a newly created real-time VM is performed in two phases. The first phase filters out a sub set of servers that can not safely accept the VM without hurting its real-time performance. In the second phase, a worst-fit algorithm is invoked to determine the best feasible server to which the real-time VM is allocated. They apply the worst-fit algorithm to achieve load balancing among the servers. A real-time version of the XEN hypervisor proposed in [33] is also applied to schedule the real-time VMs allocated to each server.

For the fifth item, a reliable solution is to establish a dedicated communication line to connect a factory to a cloud data center without any interfering with other customers of the cloud data center. This dedicated communication line could be provided either by the cloud provider or by ICT provider companies. As an alternative, the Resource Reservation Protocol (RSVP) can be used which is part of the Internet Integrated Service (IIS) model, providing real-time communication on the Internet. With RSVP, people who want to receive a real-time data stream (example, multimedia applications) can reserve a particular bandwidth in advance. Although RSVP is a more flexible and cost efficient approach to make the connection between a factory and a data-center, it is mainly suitable for soft real-time applications rather than hard real-time applications. Therefore, in our framework we suggest to use dedicated communication lines to make the bridge between the factory and a cloud data center.

For the sixth item, the time-predictable design of cloud data centers proposed in [6], [7] can be applied directly. Today's cloud computing systems neither provide a guarantee for hard real-time applications, nor provide a possibility for users to specify the deadline of their applications. Basically, these papers introduce a real-time cloud resource interface to enable users to specify the minimum amount of computing resources required to satisfy the deadline of real-time applications. They also present a resource scheduling mechanism in a cloud data-center, which is capable of guaranteeing all the deadline requirements within a data-center.

IV. Problem Modeling

Application model. We assume that the system is composed of M industrial applications $A = \{A_i : i = 1,2,...,M\}$. Each industrial application A_i consists of m_i real-time tasks $\Gamma_i = \{\tau_j^i : j = 1,2,...,m_i\}$ where i and j denote the application and task indices respectively. A task is activated periodically with period P_j^i, and deadline D_j^i. We assume four types of tasks:

1) The tasks that need an input data from a sensor located on the edge device to start their execution. Indeed, we use read-execute-write semantics which is a common strategy in several automation applications (e.g., AUTOSAR-based applications). This means, even when a new instance of a task is activated, its execution time should be delayed until receiving the input data. The size of the input data at task τ_j^i is denoted by I_j^i. The waiting time to receive the input data, if the task is executed locally on the edge node, is assumed negligible (equal to zero). Otherwise, if it is executed either on a fog node or in the cloud, it is equal to the communication network delay. The maximum communication delay to send a data from node p to node q is denoted by $d_{p,q}^{comm}(DataSize)$. To reflect the input delay in the task model, we use a release jitter denoted by J_j^i which is set equal to $d_{p,q}^{commu}(I_j^i)$. Additionally, the deadline of this type of task is set equal to its period.
2) The tasks that produce an output data at the end of their execution for an actuator located on an edge node. Let O_j^i denotes the size of output data at task τ_j^i. For such type of tasks, the generated data should be received at the actuator no later than a specific deadline which is assumed to be equal to the period of the task. Hence, the completion time τ_j^i should be less than a partial deadline which is equal to P_j^i *−the maximum communication delay to send the output*. Accordingly, the deadline of such a type of task is set to $P_j^i - d_{p,q}^{comm}(O_j^i)$. We do not need a release jitter for this type of tasks, i.e. $J_j^i = 0$.
3) The tasks that needs an input and generate an output. For this type of task, the deadline is set to $P_j^i - d_{p,q}^{commu}(O_j^i)$, and the jitter is set to $d_{p,q}^{comm}(I_j^i)$.
4) The tasks that do not send or receive any data from sensors and actuators directly. For such a type of task, the release jitter is set to zero while the deadline is set to the period of the task.

Additionally, each task has also another property specifying the execution time of the task, denoted by $C_{j,p}^i$ where p implies a processor on which the task is executed. Since we have a set of heterogeneous machines to execute a task, the execution time of a task varies on different machines. Furthermore, the memory requirement of application A_i is equal to the sum of the memory requirements of its tasks.

In such applications, the main challenge is inherent in the scheduling of the tasks of applications such that the tasks meet their deadlines while other system requirements are satisfied. A real-time application is partitioned into two subsets Γ_i^{local} and $\Gamma_i^{outsource}$ ($\Gamma_i = \Gamma_i^{local} \cup \Gamma_i^{outsource}$). Γ_i^{local} denotes the subset of tasks executed locally on the embedded device (initializing the application and hosting sensor and actuators) and $\Gamma_i^{outsource}$ indicates the subset of tasks outsourced to either fog nodes or to the cloud. The outsourced tasks are mapped to a VM to be able to execute on the fog and cloud. We use an 1-to-1 mapping of the outsourced part of the application to VM. It is a simple and efficient mapping. If we break down $\Gamma_i^{outsource}$

among multiple VMs (i.e., 1-to-N mapping), the communication overhead between those VMs is imposed on the system. On the other hand, if we place multiple applications on the same VM (i.e., N-to-1 mapping), the schedulability of the system is lowered [7]. Hence, the 1-to-1 mapping is adopted in our model. The scheduling of tasks within VMs is performed by the guest operating system.

When we partition an application into two groups we would have a communication between the nodes hosting the VM (corresponding to $\Gamma_i^{outsource}$) and the edge device hosting Γ_i^{local}. The reason of this communication comes from (i) getting the input data from sensors located on the edge device (or somewhere nearby), and (ii) sending the data to the actuators located on the edge device. The total communication data sent and received by all tasks of the ith application is denoted by CR_i. Indeed, CR_i is the important factor demonstrating whether an application is communication-intensive or not. It should be mentioned that we assume no communication between VMs and also no communication between a VM and other edge devices excluding the edge node hosing Γ_i^{local}.

From application to VM specification. We use the gEDF scheduler at the guest OS level for scheduling of tasks within VMs (task-scheduling), similar to [7]. gEDF works as follows. At each scheduling instance the scheduler selects a task with the earliest deadline and it assigns it on an idle processor. Since the task-scheduling is performed within VMs, the scheduler selects an idle virtual processor at scheduling points.

We are interested in deriving the specifications of VMs given the specification of the application that is assigned to the VM. We use the analysis framework proposed by Easwaran et. al [9] for calculating the VM specifications which is called Compositional Schedulability Analysis (CSA).

The result of the CSA analysis is the following parameters: (i) the period of the VMs denoted by Π, (ii) the total budget that has to be provided within each period (Θ) to the VMs, (iii) the maximum number of processors that can contribute in providing the total budget m' (the minimum number of processors that has to be allocated to a VM is ($\lceil \frac{\Theta}{\Pi} \rceil$)). This specification means that the underlying virtualization mechanism has to make sure that the corresponding VM receives Θ units of processor time every Π time units using $\lceil \frac{\Theta}{\Pi} \rceil$ to m' physical processors. Such a mechanism is supported by the RT-Xen hypervisor [33]. The processor utilization of each VM is defined as follows: $u = \frac{\Theta}{\Pi}$. The memory demand of a VM is denoted by h_i which is equal to the sum of the application memory requirement and the memory demand of the guest OS running within the VM. In summary, the specifications of the i^{th} VM is represented using the following tuple: $< \Pi_i, u_i, m'_i, h_i >$.

In CSA, the VM specification generation process starts by assuming a period for the VM. The period of the task with the shortest period is often selected as the period of VMs. The parallelism level m' is then selected using a binary search. For each value of m' the smallest budget is selected such that the schedulability of the task within the VM is preserved. Finally, the most efficient interface, i.e, the one with the lowest u, is selected as the VM specification.

It is worth noting that we also use a gEDF scheduler within each edge node to schedule Γ_i^{local}.

Fog model. In order to introduce a cost-effective industrial cloud framework in terms of the TCO, a fog platform is preferably developed with the minimum processing power and the least data storage size to run only the necessary portion of the workload that must be executed locally. It should be mentioned that minimizing the size of the fog does not imply to not use the maximum processing capacity of the available fog nodes (excluding local servers), instead we mean to minimize the use of the local servers.

As we have focused on real-time applications, the necessary portion of a workload is defined as a subset of applications with a tight range of latency requirements, that in order to meet their latency requirements, cannot be allocated on a cloud data center. In other words, our design strategy is to first use the available fog nodes and then, instead of using local servers, push as much as possible of the workload onto the cloud data center rather than local servers.

A fog is constructed by a set of computing and communication resources such as local servers, the remaining capacity of network switches, sinks, gateways, SDN controllers, etc. To provide an abstraction of the resources, we can consider them as a set of heterogeneous devices capable of providing computing resources to execute a VM. Therefore, a fog can be modeled as a set of $N = N^{fog} + N^{localServers}$ local devices with different processing power, memory capacity and available resources. Each device i applies either a single-core or multi-core processor. For generalization of the model, we assume a multi-core processor with $m_i \geq 1$ processing cores where $\{U_{i,1}, U_{i,2}, ..., U_{i,m_i}\}$ percent of its processors are available to use whilst the other $(1 - U_{i,j})\%$ is already dedicated to execute the primary tasks of the device. For example, if we talk about an Ethernet switch, then the primary task is to forward the network messages. Moreover, let's assume, the multiple cores of the same device have identical processing power (homogeneous multi-core). The available memory capacity of each device is denoted by H.

Since the cloud data center is highly powerful and scalable, the communication times of the VMs are the main bottleneck. Accordingly, computation-intensive applications are preferred to be executed in the cloud rather than communication-intensive applications.

We assume that the main goal is to minimize the number of used local servers. It provides two main benefits (i) most of the workload can be outsourced to the cloud which in this case both deployment cost and maintenance cost can be reduced (ii) a more available computing and storage capacity would be available on local servers that then can be consumed to run highly sensitive and confidential data and processing.

Cloud data center model. We assume a huge cloud data center hosting a large number of servers with an enormous memory and storage capacity. For simplification we can assume that the processing capacity of the cloud data center is infinite, in other words, it can always provide the demanded computation and communication resources. We also imagine the whole cloud data center as a single powerful node, and for

simplification of the problem, we assume that the execution time of each task in the cloud is known in advance, thereby we can calculate the processing utilization required to execute a VM on a cloud server.

Communication model. In this paper we use an abstracted communication model where we assume that a dedicated bandwidth is reserved to communicate between each pair of nodes (including edge nodes, fog nodes and the cloud) in the system, in the sense that the maximum communication delay to send one unit of data from node p to node q is known in advance. This can be fulfilled by Time-Division Multiple Access (TDMA) or other real-time network mechanisms. However, to have a more sensible model, we restrict the maximum communication rate on each path. In other words, the allocation of VMs to nodes should be performed such that the total rate of communication data on each path should not exceed the specified maximum threshold, otherwise the maximum communication delay is not guaranteed anymore.

The maximum threshold for the path between node p and node q is denoted by $L_{p,q}$, and the guaranteed bandwidth of this path is indicated by $BW_{p,q}$. It is worth noting that $Path_{p,q}$ may share multiple links with other paths between other pair of nodes and the guaranteed bandwidth of the path is specified according to the minimum guaranteed bandwidth of its links. Accordingly, as long as the total communication rate on $Path_{p,q}$ is less than $L_{p,q}$, the maximum communication delay to send one unit data from node p to node q is equal to $\frac{1}{BW_{p,q}}$.

V. Resource Provisioning Mechanism

As the task allocation and VM placement problem in such a diverse configuration of nodes and links is an NP-Hard problem, finding an exact solution needs an exhaustive search which is dramatically time-consuming and can not be applied for medium or large instances of the problem. Consequently, we propose a heuristic algorithm that can find a near-optimal solution in a reasonable time. This algorithm first partition each application into two groups where the first group is located on the embedded device originating the application while the second group (if it is not empty) is mapped to a single VM which then can be executed either on a fog node or in the cloud. For partitioning of an application, we use an algorithm called Lowest-Laxity First (LLF). The laxity is a metric showing the tightness of the deadline of each task, that is calculated as follows.

$$\begin{aligned} Lax_j &= P_j - \left(\frac{I_j}{BW_{min}} + \max_{\forall fogNodes\ q}(C_{j,q}) + \frac{O_j}{BW_{min}}\right) \\ BW_{min} &= \min_{\forall fogNodes\ q}(BW_{p,q}) \end{aligned} \tag{1}$$

where BW_{min} implies a fog node with the lowest bandwidth to the edge node originating the application (node p). Indeed it is used to calculate the maximum communication delay for the input data of the jth task. Similarly, the last term of this equation calculates the maximum communication delay for the output data of the jth task. The middle term denotes the execution time of the jth task on the weakest fog node on which the execution of the jth task takes longer than on other fog nodes.

The LLF partitioning algorithms first sort all the tasks of an application (ith application) according to their laxity in the ascending manner. Then it picks the first task from the sorted list and places it on the edge device originating the application. If there are enough resources on the edge node to run this application in a timely manner, then the task is added to Γ_i^{local}, otherwise it is added to $\Gamma_i^{outsource}$. To examine whether a task is able to be executed on the edge node, (i) gEDF is used to see the task is able to meet its deadline, and (ii) the total memory demand of tasks assigned on the node along with memory demand of the new task should be less than the total available memory on the edge node. LLF then goes for the next task and it continues until all tasks are partitioned into the subsets Γ_i^{local} and $\Gamma_i^{outsource}$.

Algorithm 1 indicates the proposed algorithm called Resource Allocation algorithm for Minimization Of the Fog size (RAMOF) that assigns a set of given applications onto the edge node, fog nodes and the cloud.

Algorithm 1 RAMOF

```
Inputs: a given set of applications A = {A_1, A_2, ..., A_M}
for each application A_i in A do
    partition A_i into two subsets Γ_i^local and Γ_i^outsource using LLF
    assign Γ_i^local to the edge node originating application A_i
    if Γ_i^outsource ≠ Null then
        assign Γ_i^outsource to VM_i
        increment N^VMs
    end if
end for
AR ← Call Fog_Node_Placement(VM_Set, N^VMs)
return the assignment AR, and N^UsedServers
```

Algorithm 2 Fog_Node_Placement

```
ξ = {all the fog nodes excluding the local servers}
Categorize the VMs into two separate groups, Hard_VMs and Soft_VMs
for each VM_i in Hard_VMs do
    node_j ← Find_first_feasible_node(ξ, VM_i)
    if node_j ≠ Null then
        assign VM_i to Node_j
    else
        assign VM_i to LocalServer_k
        increment N^UsedServers
        ξ ← ξ ∪ LocalServer_k
    end if
end for
Update AR by Calling Cloud_Fog_Placement(Soft_VMs, ξ)
```

If an application is able to be executed in the cloud, it is dubbed as a soft VM, otherwise it is a hard VM. To check whether an application is able to be executed in the cloud or not, all of its tasks will be examined. If the following condition holds for all tasks of the application, then it can be executed in the cloud.

$$d_{q,cloud}^{commu}(I_i) + C_{i,cloud} + d_{q,cloud}^{commu}(O_i) \leq P_i \tag{2}$$

where q is the edge node originating the application.

Algorithm 3 Cloud_Fog_Placement

```
1: sort soft_VMs according to CR_i in a descending manner
2: γ ← soft_VMs
3: for each VM_i in Soft_VMs do
4:    node_j ← Find_first_feasible_node(ξ, VM_i)
5:    if node_j ≠ Null then
6:       assign VM_i to Node_j
7:       γ ← γ − VM_i
8:    end if
9: end for
10: if γ ≠ Null {There is at least one soft VM not fitted in the available fog nodes ξ} then
11:    sort the elements of γ according to CR_i in an ascending manner
12:    for each VM_k in γ do
13:       node_j ← Find_first_feasible_node(ξ, VM_i)
14:       if the total communication data on the link to the cloud + CR_k < Max_Load_CloudLink then
15:          assign VM_k to the cloud
16:          γ ← γ − {VM_k}
17:       else
18:          break {Goto out of the loop}
19:       end if
20:    end for
21:    assign all the remaining elements of γ to the local servers, and update N^UsedServers
22: end if
23: return AR, and N^UsedServers
```

In Algorithm 3, CR_i denotes the total communication size, sending and receiving by all the tasks of the ith VM, and $Max\_Load_{CloudLink}$ implies the maximum tolerable traffic on the link to the cloud such that the maximum communication delay is respected. Furthermore, γ keeps the set of unassigned soft VMs, and whenever a VM is assigned to a node, it is excluded from this set.

VI. Performance Evaluation

By developing the code of the method proposed in Section V, we provide a software tool assisting system designers to make important decisions such as the minimum number of required local servers, number of VMs, the size of bandwidth of the links connecting the factory to the cloud. In this section, we also discuss the impact of each system parameter on the number of used local servers and their average utilization through extensive experiments.

System configuration. We consider 18 different configurations of the system including (i) various number of embedded devices (in the range of 60 to 200), (ii) different size of the bandwidth of the network connecting the factory to the cloud (in the range of 100MB/s to 2GB/s) which is shared among all the applications on the cloud using a TDMA protocol, and (iii) different processing capacity of edge nodes in terms of the number of cores of the embedded processor (in the range of 1 to 8 cores). Furthermore, in all experiments, the memory capacity of embedded devices and fog nodes is a random value (assuming uniform distribution) from [10MB, 1GB] [34], and the number of cores of the fog nodes (excluding the local servers) is a random value from [1,2], while the available capacity of each core of the fog node is assumed randomly in the range of [20% to 80%]. The number of fog nodes (excluding local servers), in all experiments, is set to 20. We also assume that the servers employ a Corei9-7980XE multi-core CPU hosting 18 processing cores with a memory of 80GB.

Application parameters. To not restrict our evaluation only for a specific automation application, we consider the range of each parameter in real-world automation applications. Then the value of each application parameter is chosen randomly (uniform distribution) from the range of the considered parameter. Accordingly, due to the considered randomness in the application parameters, different runs of the algorithm, even for the same configuration of the system, may generate different results. Hence, for each configuration of the system, we run the algorithm more than 30 times to have a comprehensive observation of the system. Then the average results of those 30 times experiments is reported here. In overall, we conduct $18 \times 30 = 540$ different experiments.

The period of the tasks are chosen from the set of periods commonly found in automation and automotive applications [1,2,5,10,20,50,100,200,1000,2000,5000]ms [35], [36] and task utilizations are generated by UUniFast [37]. The total data size of an application is considered in the range of [10,1000]KByte [38]. The number of tasks per application is assumed in the range of [10,40].

A. Results

The algorithms are implemented in C and all the experiments were executed on an Amazon EC2 Cloud virtual machine (c3.8XLarge) with 32 cores and 60GB memory, on which an Ubuntu Server 16.04 LTS instance is running.

The major metrics investigated in the evaluation of the results include the number of applications that completely fit on embedded devices, the total number of VMs, the number of VMs placed on fog nodes, the number of VMs assigned to the cloud, the number of used servers, and the average utilization of the used servers.

The results are represented in Table I where the first column indicates the number of real-time applications. As we assume that each edge node (embedded device) includes only one application, the number of applications are assumed equal to the number of edge nodes. Therefore, when we have N applications, we mean a system with N edge node. The 'Apps on edge nodes' column implies the number of applications which completely fit on the edge node, meaning that no one of its tasks is outsourced. The number of VMs reveals the number of applications that do not completely fit on the edge nodes, thus, a subset of their tasks are outsourced to fog/cloud. Moreover, the 'Hard-VMs' column of the table shows the number of VMs which are not allowed to be allocated to the cloud because of the tightness of their included tasks sets. Apparently, No. of VMs - No. of Hard VMs = No. of Soft VMs which are allowed to be placed on both cloud and fog nodes. Note that the 'Apps on fog nodes' column contains the number of applications placed on both the used local servers and other fog nodes. The last column of the table demonstrates the average execution time of the proposed algorithm. As is

TABLE I
EXPERIMENT RESULTS FOR DIFFERENT NUMBER OF REAL-TIME APPLICATIONS

Apps	VMs	Hard-VMs	Apps on edge nodes	Apps on fog nodes	Apps on the cloud	Used local servers	Avg. Util. of servers	Execution Time (sec)
60	48	15	12	38	10	4	0.759	6
80	59	19	21	45	14	6	0.844	8
100	80	26	20	63	17	7	0.851	15
120	97	29	23	77	20	8	0.835	29
140	110	33	30	82	28	8	0.895	58
160	105	34	55	83	22	9	0.846	80
180	116	35	64	85	31	9	0.874	116
200	127	39	73	99	28	10	0.865	168

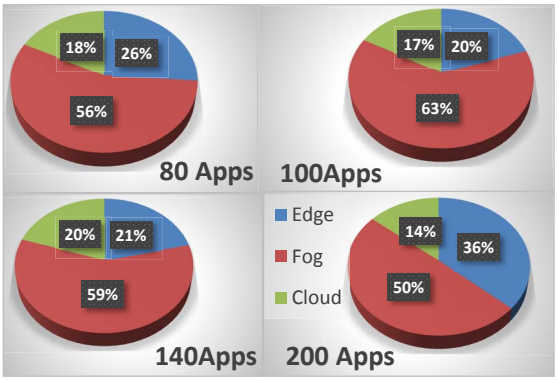

Fig. 2. Portion of applications on edge, fog and cloud.

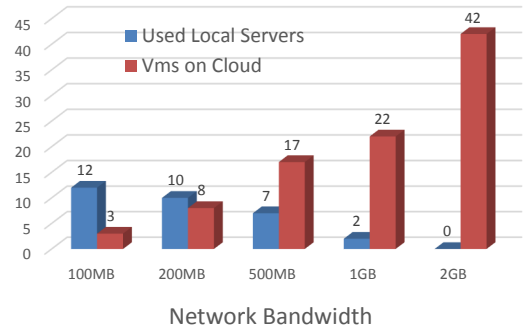

Fig. 3. Effect of the bandwidth of the network connecting the factory to the cloud on the placement.

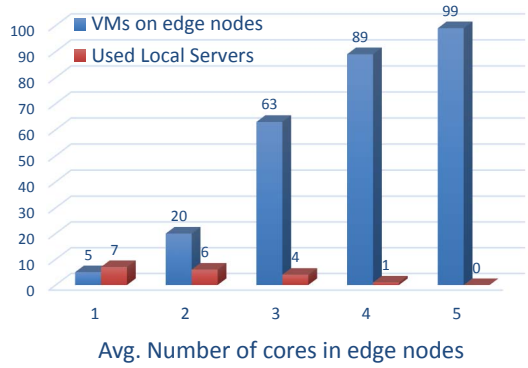

Fig. 4. Effect of the processing power of the edge nodes on the placement.

shown by this column, for a system with 200 edge nodes, the software tool can generate the results in less than 2.8 minutes, and as it is going to be used in the design time of the system, it could be a reasonable time from the system designers' perspective. Fig. 2 shows the percentage of the applications assigned to each computing layer including edge, fog and cloud. It should be mentioned that the edge portion of the pie charts implies those applications that completely fit on the edge nodes and no one their tasks are outsourced to fog or cloud.

To investigate the effect of the bandwidth of the network connecting the smart factory to the cloud on (i) the number of VMs outsourced to the cloud, and (ii) the number of used local servers, we consider different values for the bandwidth and observe its effect on these parameters. This feature can significantly help system designers to chose a proper bandwidth for the network to connect the factory to the cloud. Fig. 3 illustrates the results for 5 different sizes of bandwidth. As we theoretically expected, by increasing the size of the bandwidth, a higher number of applications can be placed on the cloud, thereby the number of utilized local servers is decreased. Therefore, increasing the bandwidth of the network connecting the factory to the cloud can reduce the size of the fog. However, the optimal value of the network bandwidth should be adopted according to the ratio of the cost of deploying fog to the cost of networking. In future work, we aim to propose a cost model for the system that can determine the optimal value of the network bandwidth and fog size according to their deployment cost.

We also conduct a set of experiments to scrutinize the impact of the processing power of edge nodes (embedded devices) on the placement desicions. We intuitively anticipate when the processing power of the edge nodes goes higher, a higher number of applications completely fit on the edge nodes, hence, a lower workload is imposed on the local servers which can result in a lower number of utilized local servers. Fig. 4 verifies this hypothesis. An interesting indication of the results is that when the average number of cores in edge nodes reaches to five, all the applications completely fit on the edge nodes, then there is no workload on the cloud and fog nodes. Here again, there is a tradeoff between the cost of increasing the processing power of embedded devices and the cost of local servers. However, determining the processing power of the embedded devices is not under the control of the IoT system designer, in other words, we assume they are already fixed. For example, when we bought a production robot arm, we do not want to change the hardware of the embedded device of the robot.

VII. Conclusion

This paper first presents a fog-integrated cloud framework for smart factories which is capable of dealing with timing requirements of real-time applications. This framework not only considers computation demand of applications but also memory requirements and communication data of applications on the network are taken into account. According to the proposed framework, we develop a software tool which enables system designers to determine the minimum number of local servers that should be integrated to the available fog nodes to provide enough resources for execution of real-time applications subject to timing and memory requirements. The proposed software tool can also guide system designers to choose a proper size of the bandwidth to link the factory to a cloud data center. For future work, we plan to consider an average active time of each application per month. Then according to this average active time, we propose an extended version of the design tool for optimizing the total cost of execution of applications.

References

[1] S. Wang *et al.*, "Implementing smart factory of industrie 4.0: An outlook," *Journal of Distributed Sensor Networks*, vol. 12, no. 1, pp. 315–329, 2016.

[2] J. Lee *et al.*, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[3] (https://www.dbresearch.com). Industry 4.0: Huge potential for value creation waiting to be tapped. deutsche bank research.

[4] S. Yin and O. Kaynak, "Big data for modern industry: Challenges and trends [point of view]," *Proc. of the IEEE*, vol. 103, no. 2, pp. 143–146, 2015.

[5] D. S. Modha *et al.*, "Cognitive computing," *Communications of the ACM*, vol. 54, no. 8, pp. 62–71, 2011.

[6] N. Khalilzad *et al.*, "Towards energy-aware placement of real-time virtual machines in a cloud data center," in *HPCC-CSS-ICESS '15*, 2015, pp. 1657–1662.

[7] S. Xi *et al.*, "RT-Open Stack: CPU resource management for real-time cloud computing," in *CLOUD*, 2015, pp. 179–186.

[8] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS'03*, 2003, pp. 2–13.

[9] A. Easwaran *et al.*, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25 –59, 2009.

[10] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *RTSS'10*, 2010, pp. 249–258.

[11] H. Leontyev and J. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," in *ECRTS'08*, 2008, pp. 191–200.

[12] A. Burmyakov *et al.*, "Compositional multiprocessor scheduling: The gmpr interface," *Real-Time Systems*, vol. 50, no. 3, pp. 342–376, 2014.

[13] S. Xi *et al.*, "Real-time multi-core virtual machine scheduling in Xen," in *EMSOFT*, 2014.

[14] H. R. Faragardi *et al.*, "Towards energy-aware resource scheduling to maximize reliability in cloud computing systems," in *HPCC*, 2013.

[15] K. H. Kim *et al.*, "Power-aware provisioning of cloud resources for real-time services," in *MGC'09*, 2009, p. 1.

[16] A. Rajabi *et al.*, "Communication-aware and energy-efficient resource provisioning for real-time cloud services," in *CADS*, 2013, pp. 125–129.

[17] A. Beloglazov *et al.*, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[18] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *MGC'10*, 2010, 4:1–4:6.

[19] J. A. Pascual *et al.*, "Towards a greener cloud infrastructure management using optimized placement policies," *Journal of Grid Computing*, pp. 1–15, 2014.

[20] R. Mahmud *et al.*, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything*, Springer, 2018, pp. 103–130.

[21] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Workshop on Mobile cloud computing*, 2012, pp. 13–16.

[22] D. Zeng *et al.*, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. on Computers*, vol. 65, no. 12, pp. 3702–3712, 2016.

[23] Z. Wen *et al.*, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017.

[24] Y. Mao *et al.*, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[25] J. Liu *et al.*, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Information Theory (ISIT), 2016 IEEE International Symposium on*, IEEE, 2016, pp. 1451–1455.

[26] M. Villari *et al.*, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[27] H. C. Foundation, "Hart communication protocol specification," *HART Communication Foundation Std. HCF_SPEC-13, Rev. 7.5*, 2013.

[28] G. Prytz, "A performance analysis of ethercat and profinet irt," in *ETFA*, 2008, pp. 408–415.

[29] (2014). What is the difference between ethernet and industrial ethernet, [Online]. Available: http://www.innovasic.com/news/industrial-ethernet/what-is-the-difference-between-ethernet-and-industrial-ethernet/.

[30] J. Song *et al.*, "WirelessHART: Applying wireless technology in real-time industrial process control," in *RTAS*, 2008, pp. 377–386.

[31] D. Dujovne *et al.*, "6tisch: Deterministic ip-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[32] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Trans. on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 30, 2008.

[33] S. Xi *et al.*, "Real-time multi-core virtual machine scheduling in xen," in *EMSOFT*, 2014, pp. 1–10.

[34] A. Yousefpour *et al.*, "Qos-aware dynamic fog service provisioning," in *WATERS*, 2015, pp. 1–6.

[35] G. Giannopoulou *et al.*, "Mapping mixed-criticality applications on multi-core architectures," in *DATE*, 2014, pp. 1–6.

[36] S. Kramer *et al.*, "Qos-aware dynamic fog service provisioning," in *Conf. on Fog and Edge Computing*, 2017, pp. 1–6.

[37] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1, pp. 129–154, 2005.

[38] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *WWC-4*, 2001, pp. 3–14.