Hamid Reza Faragardi received his BSc degree in Computer Eng. in 2010, followed by a MSc in Computer Eng. from University of Tehran in 2012, and a Licentiate degree in Computer Engineering from Mälardalen University, Sweden in 2017. From January 2018, he started a postdoc at Distributed and Parallel System (DPS) Group at Innsbruck University in Austria. He is also working as a research engineer at KTH Royal Institute of Technology in Stockholm where he is supervising multiple Master students. His main research interests comprise cloud computing, real-time scheduling, Industrial IoT, and optimization problems.
OPTIMIZING TIMING-CRITICAL CLOUD RESOURCES IN A SMART FACTORY

Hamid Reza Faragardi

2018
OPTIMIZING TIMING-CRITICAL CLOUD RESOURCES IN A SMART FACTORY

Hamid Reza Faragardi

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid
Akademin för innovation, design och teknik kommer att offentligen försvaras
 torsdagen den 8 mars 2018, 13.30 i Gamma, Mälardalens högskola, Västerås.

Fakultetsopponent: Professor Ivona Brandic, Technische Universität Wien

Akademin för innovation, design och teknik
Abstract

This thesis addresses the topic of resource efficiency in the context of timing critical components that are used in the realization of a Smart Factory. The concept of the smart factory is a recent paradigm to build future production systems in a way that is both smarter and more flexible. When it comes to realization of a smart factory, three principal elements play a significant role, namely Embedded Systems, Internet of Things (IoT) and Cloud Computing. In a smart factory, efficient use of computing and communication resources is a prerequisite not only to obtain a desirable performance for running industrial applications, but also to minimize the deployment cost of the system in terms of the size and number of resources that are required to run industrial applications with an acceptable level of performance. Most industrial applications that are involved in smart factories, e.g., automation and manufacturing applications, are subject to a set of strict timing constraints that must be met for the applications to operate properly. Such applications, including underlying hardware and software components that are used to run the application, constitute a real-time system. In real-time systems, the first and major concern of the system designer is to provide a solution where all timing constraints are met. To do so we need a time-predictable IoT/Cloud Computing framework to deal with the real-time constraints that are inherent in industrial applications running in a smart factory. Afterwards, with respect to the time predictable framework, the number of required computing and communication resources can and should be optimized such that the deployed system is cost efficient. In this thesis, to investigate and present solutions that provide and improve the resource efficiency of computing and communication resources in a smart factory, we conduct research following three themes: (i) multi-core embedded processors, which are the key element in terms of computing components embedded in the machinery of a smart factory, (ii) cloud computing data centers, as the supplier of a massive data storage and a large computational power, and (iii) IoT, for providing the interconnection of computing components embedded in the objects of a smart factory. Each of these themes are targeted separately to optimize resource efficiency. For each theme, we identify key challenges when it comes to achieving a resource-efficient design of the system. We then formulate the problem and propose solutions to optimize the resource efficiency of the system, while satisfying all timing constraints reflected in the model. We then propose a comprehensive resource allocation mechanism to optimize the resource efficiency in the whole system while considering the characteristics of each of these research themes. The experimental results indicate a clear improvement when it comes to timing-critical IoT / Cloud Computing resources in a smart factory. At the level of multi-core embedded devices, the total CPU usage of a quad-core processor is shown to be improved by 11.2%. At the level of Cloud Computing, the number of cloud servers that are required to execute a given set of real-time applications is shown to be reduced by 25.5%. In terms of network components that are used to collect sensor data, our proposed approach reduces the total deployment cost of the system by 24%. In summary these results all contribute towards the realization of a future smart factory.
Denna avhandling behandlar ämnet resurseffektivitet i samband med tidskritiska komponenter som används vid förverkligandet av en Smart Fabrik.


Den här avhandlingen undersöker och presenterar lösningar som förbättrar
Abstract

This thesis addresses the topic of resource efficiency in the context of timing-critical components that are used in the realization of a Smart Factory.

The concept of the smart factory is a recent paradigm to build future production systems in a way that is both smarter and more flexible. When it comes to realization of a smart factory, three principal elements play a significant role, namely Embedded Systems, Internet of Things (IoT) and Cloud Computing. In a smart factory, efficient use of computing and communication resources is a prerequisite not only to obtain a desirable performance for running industrial applications, but also to minimize the deployment cost of the system in terms of the size and number of resources that are required to run industrial applications with an acceptable level of performance.

Most industrial applications that are involved in smart factories, e.g., automation and manufacturing applications, are subject to a set of strict timing constraints that must be met for the applications to operate properly. Such applications, including underlying hardware and software components that are used to run the application, constitute a real-time system. In real-time systems, the first and major concern of the system designer is to provide a solution where all timing constraints are met. To do so we need a time-predictable IoT/Cloud Computing framework to deal with the real-time constraints that are inherent in industrial applications running in a smart factory. Afterwards, with respect to the time predictable framework, the number of required computing and communication resources can and should be optimized such that the deployed system is cost efficient.

In this thesis, to investigate and present solutions that provide and improve the resource efficiency of computing and communication resources in a smart factory, we conduct research following three themes: (i) multi-core embedded processors, which are the key element in terms of computing components embedded in the machinery of a smart factory, (ii) cloud computing data centers,
as the supplier of a massive data storage and a large computational power, and
(iii) IoT, for providing the interconnection of computing components embed-
ded in the objects of a smart factory. Each of these themes are targeted sepa-
rately to optimize resource efficiency. For each theme, we identify key chal-
lenges when it comes to achieving a resource-efficient design of the system.
We then formulate the problem and propose solutions to optimize the resource
efficiency of the system, while satisfying all timing constraints reflected in the
model. We then propose a comprehensive resource allocation mechanism to
optimize the resource efficiency in the whole system while considering the
characteristics of each of these research themes. The experimental results in-
dicate a clear improvement when it comes to timing-critical IoT / Cloud Com-
puting resources in a smart factory. At the level of multi-core embedded de-
VICES, the total CPU usage of a quad-core processor is shown to be improved
by 11.2%. At the level of Cloud Computing, the number of cloud servers that
are required to execute a given set of real-time applications is shown to be
reduced by 25.5%. In terms of network components that are used to collect
sensor data, our proposed approach reduces the total deployment cost of the
system by 24%. In summary these results all contribute towards the realization
of a future smart factory.
To my dear family.
Acknowledgment

First of all I would like to give thanks to my supervisors Prof. Thomas Nolte, Prof. Björn Lisper, Prof. Kristian Sandström, and Dr. Alessandro Papadopoulos. Fulfilling this goal would not have been possible without their kind support, encouragement and valuable comments.

I would also like to thank my office mate Dr. Matthias Becker for his help and comments on my research project.

A special thanks goes to the CORE research group for providing a friendly environment, specifically to: Prof. Thomas Nolte, Associate Prof. Moris Behnam, Prof. Kristian Sandström, Dr. Alessandro Papadopoulos, Dr. Matthias Becker, Dr. Mohammad Ashjaei, Dr. Saad Mubeen, Dr. Sara Afshar.

I also would like to appreciate the IDT administration staff for their help to deal with all practical issues. Many thanks go to Carola, Jenny, Sofia and the others.

I would also like to thank my friends and colleagues at the department for all the fun we have during the work and conference trips: Matthias, Dag, Arash, Hossein, Farid, Sara Ab., Mohammad, Alessandro, Sara x. Ab., Mehrdad, Sahar, Meng, Nesredin, Nima, Saad, Sara Afshar and others!

Last, I would like to express my gratitude to my family for the constant support and love throughout my life.

This work has been supported by Mälardalen University, Vinnova via the FFI initiative “AUTOSAR for Multicore in Automotive and Automation Industries”, and KKS via the initiative PREMISE “Predictable Multicore Systems”.

Hamid Reza Faragardi
Västerås, January 10, 2018
List of publications

Papers included in the PhD thesis


---

The included articles have been reformatted to comply with the PhD thesis layout.
Additional papers, not included in the PhD thesis


## Contents

I Thesis 1

1 Introduction 3
   1.1 Motivation 3
   1.1.1 Research Themes 5
   1.2 Outline of the Thesis 10

2 Background 11
   2.1 Embedded Systems 12
      2.1.1 Real-Time Systems 13
      2.1.2 Scheduling 13
   2.2 Cloud Computing 15
      2.2.1 Cloud Data Center 16
      2.2.2 Energy Consumption of a Data Center 18
   2.3 Industrial IoT 21
      2.3.1 Fog Computing 23
      2.3.2 Computational Offloading 24
      2.3.3 Industrial Software Architectures 24

3 Research Summary 27
   3.1 Problem Statement and Research Goals 27
   3.2 Research Challenges 27
      3.2.1 Research Challenge 1 (RC1) 27
      3.2.2 Research Challenge 2 (RC2) 28
      3.2.3 Research Challenge 3 (RC3) 28
      3.2.4 Research Challenge 4 (RC4) 29
   3.3 Research Questions 30
   3.4 Research Methodology 30
3.5 Validity of the Results ........................................... 32
  3.5.1 Construct Validity ........................................ 32
  3.5.2 Internal Validity .......................................... 33
  3.5.3 Conclusion Validity ...................................... 33
3.6 Discussion Regarding Sensitivity Analysis of Parameters . . . . 34

4 Contributions and Discussion .................................. 37
  4.1 Overview of the Papers ..................................... 37
  4.2 Personal Contribution ....................................... 41

5 Conclusions and Future Work ................................ 45
  5.1 Summary .................................................... 45
  5.2 Future Work ................................................. 47

Bibliography ....................................................... 49

II Included Papers ............................................... 57

6 Paper A: A Resource Efficient Framework to Run Automotive Embedded
  Software on Multi-core ECUs .................................. 59
  6.1 Introduction ................................................ 61
  6.2 Related Work ............................................. 64
  6.3 Problem Modeling .......................................... 66
     6.3.1 The AUTOSAR Architecture ............................. 66
     6.3.2 Problem Description .................................. 68
     6.3.3 Communication Time Analysis ........................ 75
     6.3.4 Formal Definition of the Problem .................... 78
     6.3.5 Optimization Problem ................................. 79
  6.4 Solution Framework ........................................ 81
     6.4.1 Method 1: Simple Mapping-based Approach ........ 82
     6.4.2 Method 2: The Feedback-based Refinement Approach
               (SMSAFR) ........................................... 87
     6.4.3 Method 3: The Utilization-based Refinement Approach
               (PUBRF) ............................................ 88
     6.4.4 Illustrative Examples ................................. 92
  6.5 Performance Evaluation .................................... 95
     6.5.1 Alternative Frameworks ............................... 96
     6.5.2 Application and Hardware Specifications ........... 98
6.5.3 Comparison Results .................................. 102
6.6 Conclusion .............................................. 106
Bibliography ................................................. 109

7 Paper B:
EAICA: An Energy-aware Resource Provisioning Algorithm for Real-Time Cloud Services

7.1 Introduction ............................................ 119
7.2 Related Work .......................................... 121
7.3 Problem Description ................................. 122
  7.3.1 Notation ............................................. 122
  7.3.2 Data Center Topology ............................ 123
  7.3.3 Service Model ..................................... 124
  7.3.4 Principle Constraints .................. 127
  7.3.5 Host Power Modeling ...................... 129
  7.3.6 Network Power Modeling ............. 130
  7.3.7 Energy Modeling .............................. 130
7.4 Optimization Problem ............................. 131
7.5 Solution Approach .................................. 132
  7.5.1 Imperialist Competitive Algorithm (ICA) ............................................. 133
  7.5.2 Online Scheduling Algorithm (EAICA) ............................................. 134
7.6 Performance Evaluation .......................... 138
7.7 Conclusion ............................................. 141
Bibliography ................................................. 143

8 Paper C:
An Energy-Aware Time-Predictable Cloud Data Center

8.1 Introduction ........................................... 151
8.2 Related Work ........................................ 153
8.3 Model ................................................ 156
8.4 Solution .............................................. 159
  8.4.1 From Application to VM Specification ...... 160
  8.4.2 VM to Core Allocation ....................... 161
  8.4.3 Placement of VMs to Servers ............. 162
  8.4.4 Illustrative Example ......................... 171
8.5 Performance Evaluation ......................... 171
  8.5.1 Simulation Setup ............................. 171
  8.5.2 Alternative Methods ....................... 174
  8.5.3 Results ........................................ 174
8.5.4 Sensitivity Analysis .............................................. 181
8.6 Conclusions and Future work ........................................ 185
Bibliography ............................................................ 187

9 Paper D:
A Cost Efficient Design of a Multi-sink Multi-controller WSN in a
Smart Factory ......................................................... 193
9.1 Introduction .......................................................... 195
9.2 Related Work ........................................................ 196
9.3 Problem Modeling ...................................................... 198
  9.3.1 Problem Description .............................................. 198
  9.3.2 Optimization Problem .......................................... 200
9.4 Solution Framework .................................................. 203
9.5 Performance Evaluation ............................................. 209
  9.5.1 R-GRASP-MSP .................................................. 211
  9.5.2 Results ........................................................ 212
9.6 Conclusion .......................................................... 214
Bibliography ............................................................ 215

10 Paper E:
A Time-Predictable Fog-Integrated Cloud Framework: One Step
Forward in the Deployment of a Smart Factory ......................... 219
10.1 Introduction .......................................................... 221
10.2 Related Work ........................................................ 224
10.3 A time-predictable cloud framework ................................ 225
10.4 Problem Modeling ...................................................... 228
10.5 Resource Provisioning Mechanism .................................. 232
10.6 Performance Evaluation ............................................. 234
  10.6.1 Results ........................................................ 236
10.7 Conclusion .......................................................... 238
Bibliography ............................................................ 241
I

Thesis
Chapter 1

Introduction

1.1 Motivation

The term *Industrie 4.0* [1] is used for the next industrial revolution – which is about to take place. It is the current trend of automation in manufacturing technologies to create what has been called a *Smart Factory* [2]. In a smart factory, objects, machines and humans are interacting intelligently to achieve a flexible and reconfigurable manufacturing system. The next industrial revolution is still in infancy, in the sense that there are still a large number of open challenges to be addressed.

In many instances of industrial applications, real-time constraints need to be interleaved in the modeling of the system. Such applications are what is called *real-time systems*. In such systems the real-time constraints could be either *hard* real-time constraints, found in e.g., the Anti Lock Braking System (ABS) of a vehicle where failure to meet the constraint could result in catastrophic consequences, or *soft* real-time constraints, e.g., constraints on the arrival of the next frame of a video in a multimedia application, which is not as time-critical as hard real-time applications and would not result in a catastrophic consequence.

In addition to real-time constraints, there could be other complex constraints concerning computing and communication resources of a smart factory, e.g., constraints on amount of memory or size of communication bandwidth.

In this work, we focus on challenges related to optimizing the resource efficiency in a smart factory, where the software is subject to real-time constraints as well as limitations concerning memory and bandwidth.
Chapter 1. Introduction

When dealing with real-time applications, the performance inherent in the execution of software may not be the major concern as long as all real-time constraints are met. In such systems, usually an additional criterion is defined as the main concern of the system designers, i.e., the resource efficiency. In this thesis, we define resource efficiency as the amount of resources that are required to execute a set of real-time applications with an acceptable performance (rather than high performance), while the application and system constraints are met. Accordingly, a highly efficient system is able to execute industrial applications using a minimum number of resources without sacrificing real-time constraints.

The reduction of the amount of required resources in a smart factory yields several advantages such as:

1. Being able to serve a bigger workload using the same number and amount of resources.
2. Minimizing the cost of deployment of the system as a result of a lower number of required resources.
3. Decreasing the operational cost including the maintenance cost and the cost of energy consumption.
4. Potential reduction of negative environmental impacts as a result of a lower energy consumption.
5. Increasing reliability and availability as a potential result of being able to execute a higher number of replicas.

In order to achieve a resource efficient design of a smart factory, each of the key technologies used when constructing such smart factory must be discussed and studied one by one. Among the technologies, three principal elements include

1. embedded systems as the local computing unit,
2. the Internet-of-Things (IoT) as the interconnect, and
3. cloud computing as the global high performance computing unit.

These three principal elements together constitute the foundation for the realization of a smart factory. Accordingly, we divide our research along three research themes corresponding to each of these three elements. Note that our
perspective focuses on computing and communication resources rather than other types of resources in a smart factory (e.g., not human resources). In the following, these three research themes are investigated in more detail.

1.1.1 Research Themes

Theme I

The first research theme concerns the “local” embedded systems, i.e., where a computer system is placed into a bigger physical/electrical system, playing a role of a controller for the bigger system. Embedded systems already play a significant role in our lives. More than 98% of all processors produced worldwide are employed into embedded devices [3], for instance, in the Anti Lock Braking System (ABS) in vehicles, in smartphones, and in ordinary household appliances such as fridges, stoves, and microwaves. Examples of embedded devices in a smart factory include Programmable Logic Control (PLC) units in a robot and, e.g., the controller of the motors driving assembly belts and other industrial production plant systems.

Theme II

The second research theme concerns cloud computing. In a smart factory, we encounter an exponential increase of data size and computational complexity in comparison to traditional manufacturing factories, because of (i) participation of a higher number of nodes, each of which can continuously generate 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] (also called mind-like computing) 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 servers, 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 economy. Hence, cloud computing is a key component in a smart factory to provide highly scalable computing and storage capacity.
Theme III

The third research theme concerns an intermediate computing layer placed in between embedded devices and a cloud data center. The IoT system used in a smart factory is dubbed as Industrial IoT (IIoT) [6]. Although IIoT significantly overlaps with the definition of general IoT, it embraces more serious concerns with regards to timeliness and reliability of the system. The reason is inherent in more strict requirements of industrial applications in comparison to other classical IoT applications such as those found in smart homes [7]. IIoT not only extends the context of Wireless Sensor Networks (WSNs) to provide a greater efficiency, flexibility and scalability (for example using Software Defined Networking (SDN) controllers to address the dynamics of the network), but it also introduces new technologies to deal with the strict requirements of industrial applications, e.g., the use of fog computing. IoT devices include sensors, actuators and a limited computation and storage capacity. The final computation often requires a massive computation resource meaning that it must be performed in a more powerful entity. If only cloud computing is considered as a solution to overcome the limited computational capacity of IoT devices, then it introduces new latency in the system that impacts the actual operation of industrial control processes, and may as a result not be a feasible solution for communication intensive applications with a tight deadline. That is the reason behind introducing fog computing as an complementary computing tier incorporating with cloud computing in IIoT. The fog computing layer is constructed by a set of fog nodes. A fog node could be either the remaining capacity of an existing network device (routers, mobile stations, gateways, SDN controllers) or it could be a local server, as well as a dedicated device [8].

The Big Picture

In this thesis, each of the three research themes I, II, and III are targeted separately to optimize the overall resource efficiency in a smart factory. Within each of the research themes, we identify key challenges to achieve a resource-efficient design of the system. We then formulate corresponding problems to be solved, that are addressed subsequently. Finally, we propose a comprehensive resource allocation mechanism to optimize the resource efficiency in the whole system while considering the identified challenges within each research theme.

Figure 1.1 shows a smart factory where the different research themes are specified. As illustrated by this figure, IIoT not only includes embedded devices, sensors, and actuators with a limited computing resources, but it also
embraces wired/wireless communication networks to inter-connect the devices (things) and also to make a bridge between devices and fog nodes. Moreover, on-device computing is referred to as the local processing performed within embedded devices. The smart factory is connected to a cloud data center through a high speed communication network capable of providing an acceptable Quality of Service (QoS). A reliable solution to implement the high-speed communication network is to establish dedicated communication lines to connect a factory to a cloud data center with no interference with other network traffic. As an alternative, the Resource Reservation Protocol (RSVP) [9] can be used to establish a real-time communication link on the Internet. This figure also represents a general picture of a cloud data center where a large number of servers are connected through a hierarchical communication network, on which cloud applications – such as OpenStack – are executed to provide a transparent pool of computing and communication resources.

Figure 1.1: An example of a smart factory.

In order to design such a complex system we first need to specify the minimum amount of computing and communication resources to satisfy all the
system and applications constraints. For example, we would like to know

- How much should the minimum processing capacity of embedded devices be set to?

- How much should the minimum bandwidth of the local area network (wired, wirelesses or combination of both) be set to, to inter-connect the embedded devices in a smart factory and also to connect them to local servers and gateways?

- How much should we dimension the minimum storage and processing capacity of the fog nodes to satisfy the timeliness, reliability and other requirements of industrial applications?

- How much should the minimum bandwidth of the links connecting the factory to a cloud data center be set to?

- How much should be the minimum storage, communication and processing capacity of a cloud data center be configured to?

One of the possible solutions to address such questions is to propose a resource allocation mechanism which consolidates a maximum workload on a minimum amount of resources. This allocation mechanism has two dimensions, horizontally and vertically. In vertical dimension we need to partition a given workload among on-device resources, fog resources and a cloud resource, providing three different computing tiers. In the horizontal dimension we have to decide about the allocation of a given workload among different elements of a considered tier. For example, if we decide to allocate a subset of a workload onto fog nodes, then how should the workload be distributed among different fog nodes? This question also holds for other tiers (i.e., on-device and cloud) when it comes to deciding about the horizontal placement. Figure 1.2 shows the placement dimensions in a smart factory.

About the horizontal placement mechanism, similar strategies can be applied to achieve a resource efficient solution for each of the tiers. Although these three research themes may seem to have totally different characteristics, there are several shared concepts and similar challenges between them. Indeed, in all of these research themes, the problem is to allocate application components (a given workload) that may communicate with each other, subject to specific deadlines, to maximize the efficiency of the system while also satisfying other application and system constraints. The approaches are also the same in principle, which begin with defining a goal function specifying the efficiency
metric, then modeling the real-time and other systems constraints, continued by presenting the problem as an optimization problem, afterward solving the optimization problem using heuristic algorithms. Finally, to evaluate the proposed method, simulation experiments are conducted where the values of parameters are derived from real-world benchmarks.

For the vertical placement mechanism the problem is a high-level-version of the same problem as the horizontal placement. Here again, we have a set of elements where each element corresponds to one of the research themes with a particular computing/communication capacity subject to timing and other application constraints (e.g., reliability constraints). Similar to the horizon-
tal placement problem, the vertical placement problem can be modeled as an optimization problem and be addressed by a heuristic algorithm.

1.2 Outline of the Thesis

This thesis is organized in 9 sections. First, Section 2 introduces the required background. Section 3 presents the research summary and the research methodology used in the thesis. Research challenges are also highlighted in this section. The contributions of the thesis along with an overview of the papers included in the thesis are presented in Section 4. Finally, in Section 5 we present the conclusions and future work.
Chapter 2

Background

In this section we provide high level background information needed to contextualize the thesis.

Looking at the scope of this thesis, traditionally, mass production has been the main strategy that is adopted by the manufacturing industry when it comes to reducing the overall cost, by making several copies of a product quickly and efficiently. Product variation (i.e., customized products) imposes down-time for resetting equipment and retooling, which decreases volumes and increases costs. In order to make variation production cost-efficient, future production lines need to be smarter and more flexible [10]. This is the principal notion of Industry 4.0 [11]. Industry 4.0 creates what has been called a Smart Factory. To achieve the goal of a smart factory, all the manufacturing processes are supposed to be configurable and connected to the Internet. The idea can be fulfilled by connecting and integrating traditional industries, and by providing communication between producers and consumers. This idea in the near future will revolutionize the whole industrial panorama as pointed out by the Fraunhofer Institute and the industry association Bitkom; “German gross value can be boosted by a cumulative 267 billion Euros by 2025 after introducing Industry 4.0” [1]. Similar strategies have also been proposed by other main industrial countries, for example, Industrial Internet [12] from USA and Made in China 2025 [13] from China. Manufacturing companies can realize and implement Industry 4.0 through the four design principles [2] as listed below.

1. Connectivity: It refers to the ability of machines, devices, sensors, and humans to be connected and able to interact with each other through the
usage of IoT. This principle also points out machine-to-machine communication where the data collected using a machine is sent to other machines to be analyzed and make decisions independent of the human decisions.

2. Virtualization of physical systems: It refers to create a virtual image of physical devices by using various types of sensors. It needs the ability of analyzing raw data of sensors to achieve a high level context information.

3. Human-Machine incorporation: It refers to the ability of physical systems to help humans by performing various tasks that are difficult, unpleasant, exhausting, or unsafe for humans.

4. Decentralized decisions: When a machine can analyze the raw data coming from other machines, then it can take decisions on its own and perform its tasks independently and autonomously. Only in the case of exceptions the human interventions may be required.

As presented in the previous chapter of this thesis, the three research themes are the central building blocks of a smart factory, namely i) embedded systems, ii) cloud computing and iii) industrial IoT. In the following, a brief background for each of these research themes is presented.

2.1 Embedded Systems

*Embedded systems* are those systems where a computer system is embedded as part of a complete device often including hardware and mechanical parts [14]. Most of embedded systems have a dedicated function, often subject to timing constraints. In such systems, due to resource limitations, efficient use of the resources such as memory, the CPU, and communication equipment, is central when it comes to the design of the system.

*Multi-core embedded systems* are embedded systems utilizing processors with multiple cores. Such processors potentially have much higher computational capacity compared to traditional single-core processors and, consequently, the manufacturing industry has started to migrate their systems and products from single-core processors to parallel multi-core processors, i.e., single-core ECUs are being replaced by multi-core ECUs. The reason behind this shift is mainly due to the tremendous growth in size, number and computational complexity of software features, emerging as a result of offering modern
facilities for customers. If more powerful processors are considered as the solution to this growth in demand, then due to longer pipelines and higher clock frequencies, timing anomalies will increase. Therefore, multi-cores are being widely touted as the effective solution, offering a better performance per watt along with a high potential for scalability [15]. Moreover, multi-core processors are efficiently able to co-host applications with different criticality levels, allowing for the co-hosting of non-safety and safety-critical applications on the same processor. Even though multi-core processors offer several benefits to embedded systems, it is more complicated to efficiently develop an embedded software on a multi-core processor than on a traditional single-core processor.

2.1.1 Real-Time Systems

A real-time system is a system for which correctness is also depending on the timing of when an operation is performed, i.e., once an embedded system includes real-time requirement(s), the correctness of a computation is not solely enough; the response time of the computation is also important. In other words, to fulfill the functionality of the system, the response should be produced within a specific time referred to as deadline. The deadlines are inherent in the application requirements, and according to the specific application requirements the deadlines could be either what is called hard or soft. For example, in the automotive industry the deadline associated to the braking system is a hard deadline, i.e., the corresponding computations absolutely need to be performed before the deadline otherwise catastrophic consequences may occur.

2.1.2 Scheduling

One of the principal solutions for system designers when it comes to satisfying real-time requirements is scheduling. The scheduler specifies the order of execution of the tasks such that they can meet their deadlines. In real-time systems, to ensure that a given task set is able to meet its deadlines, two offline operations are performed,

1. worst case execution time analysis, and
2. schedulability analysis.

In the former phase, the worst case execution time of tasks is calculated, while in the latter phase, the worst-case timing behavior of the system under a particular scheduling mechanism is examined, and it produces a yes/no answer for the schedulability of the considered task set.
Fixed Priority Scheduling (FPS) is a well-known scheduling mechanism employed in a large number of real-time systems. The schedulability analysis of FPS was proposed in [16]. Earliest Deadline First (EDF) is another well-known scheduling algorithm, in which, as far as the total utilization of the given workload is not greater than one, all the tasks of the workload meet their deadlines [17].

Concerning multi-core scheduling there are two main approaches that provide both advantages and disadvantages:

1. **Partitioned scheduling** where the given task set firstly is divided among processors (i.e., task allocation), afterward, the subset of tasks dedicated to each processor is scheduled by a uniprocessor scheduling algorithm such as FPS or EDF [18], as depicted in Figure 2.1.

![Figure 2.1: Partitioned scheduling scheme.](image)

Partitioned scheduling enables us to employ well-known uniprocessor schedulers along with their schedulability analysis. Additionally, partitioned scheduling has a low run-time complexity, since task migration is not allowed. Nevertheless, it adds a new challenge to the system design i.e., the task allocation problem. If a given task set is not properly partitioned among different processors, it is not possible to efficiently utilize the processors. Accordingly, an optimal task allocation is key to achieve good resource efficiency of multi-processor systems under the partitioned scheduling. Unfortunately, the task allocation problem is known to be an NP-Hard problem [19], thus, finding the optimal allocation of tasks to processors can not be achieved in a non-exponential
2.2 Cloud Computing

(i.e., polynomial) time. Accordingly, a large number of heuristic algo-
rithms have been introduced [20, 21, 22] to find near-optimal solutions
for allocation of real-time tasks to processors in a polynomial time.

2. Global scheduling where a single shared queue is employed to sched-
ule the tasks instead of multiple dedicated queues as is used in parti-
tioned scheduling [18]. In this case, multiple jobs of a task may execute
on different processors. Global EDF is one of the well-known global
schedulers which however is not optimal, i.e., if a task set is not schedu-
table under the Global EDF, it may be schedulable under other algo-
rithms. Proportionate fair (pfair) is one of the global scheduling algo-
rithms which is optimal for periodic tasks with implicit deadlines [23],
however, it requires many preemptions and migrations which yields a
considerable overhead for this algorithm. The concept of global schedul-
ing is depicted in Figure 2.2.

Figure 2.2: Global scheduling scheme.

2.2 Cloud Computing

Traditionally, enterprises have been buying their own computing infrastruc-
tures to run their applications, such as financial analysis, distributed data pro-
cessing, etc. However, the cloud computing model brought a highly scalable
solution for various types of user demands. Indeed, cloud computing is a new
generation of distributed systems inheriting several properties and paradigms
from previous computing models, such as mainframe computing, cluster com-
puting and grid computing.

Looking at the evolution of cloud computing, its notion is inspired by the
mainframe computing model in which multiple users are able to share a pow-
erful computer (at that time) and accessing this resource through terminals.
Basically, by sharing computing resources, the mainframe was providing a both cost-effective and practical solution. Furthermore, grid computing has a wide range of similarities with cloud computing, whereas the major differences come from two prominent features including utility computing [24] and software services, which are presented in the context of cloud computing. The obstacles and opportunities of cloud computing are discussed in [25].

Cloud computing services are categorized in three different service models: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). In the SaaS form, a cloud provider offers a software application as an on-demand service, then users can use services through a subscription, following a “pay-as-you-go” model. Dropbox, Google Docs, and Office 365 are examples of this form of cloud services. In the PaaS model, a platform for developing web applications is provided, and developers can gain from the in-place facilities to quickly create their own applications. PaaS makes the development, testing, and deployment of applications simple and cost-effective. Apprenda, Google App Engine, Heroku are examples of this form. In the IaaS form, computing and communication hardware resources (such as servers, storage, network and operating systems) are provided as on-demand services following the pay-as-you-go model [26]. In this self-service model, users can create their own Virtual Machines (VMs) and specify the required processing power of the VMs and networking services (e.g. firewalls). Amazon Web Services EC2, Rackspace, Google Compute Engine are examples of this category of cloud services. Figure 2.3 depicts these three categories in a cloud computing system. In this work however, we mainly focus on SaaS and IaaS.

2.2.1 Cloud Data Center

Cloud computing is performed utilizing computational capacity provided by a cloud data center. A cloud data center contains a set of servers (hosts) connected together through a communication network. The servers in a data-center could be identical which is called a homogeneous data-center, or they could have different characteristics which is called a heterogeneous data-center. Each server itself consists of multiple components such as multi-core processors, disk storage, memory modules and I/O devices. Figure 2.4 represents an abstract image of a cloud data center.
2.2 Cloud Computing

There are different commonly used network architectures employed within the data center to inter-connect cloud servers. Among these network architectures the two-tier and three-tier trees of servers and switches are the most popular network architectures used within cloud data centers [27]. A three-tier network is shown in Figure 2.5. The first tier is the core part of the network located at the root of the tree. The switches at this level are used only for inter-module communication (as is shown in the figure, a module comprises a set of racks). The second tier is an aggregation network which provides both routing and intra-module communication. At the bottom of this tree, the access tier is located holding the pool of hosts. The switches at this level are used to conduct communication between different servers of the same rack. The routing mechanism in such networks is usually performed dynamically with respect to feedback from the load of switches (adaptive routing). Therefore, to send data from one host to another, more than one particular route may be adopted at run-time. Older data centers often used a two-tier architecture in which there
is no aggregation tier. Due to the scalability limitation of the two-tier network, it is not suitable for today’s data centers where the number of hosts typically exceeds 100,000 [28].

2.2.2 Energy Consumption of a Data Center

In large data centers the energy consumption plays an important role. Energy consumption of data centers is around 1.5% - 2% of the total energy consumption in the US, while this is around 1.5% in the world [29, 30]. Three main elements contribute to the energy consumption of a data-center,

- cooling and power distribution system,
- computing servers, and
- network equipment.

In this thesis the energy consumption of servers and the communication network are mainly targeted. One of the most efficient approaches to reduce the energy consumption of network devices is to dynamically put some unused switches in sleep mode (or to turn them off). It should be emphasized that it is not recommended to put the switches of the core tier in sleep mode. The reason is that keeping the core switches turned on improves the network reliability. Therefore, in this thesis, when we are targeting energy consumption, we will focus only on the aggregation (module) and access (rack) switches.
Power Consumption of Servers

The power consumption of a server consists of the collective power consumed by CPU, memory, disk storage and network interfaces. It is shown by [31] that the power consumption of the CPU is one of the main factors in the overall power consumption of a host. Accordingly, the power consumption of a host can be modeled based on the CPU utilization by the usage of a linear model [32] as defined in Eq. 2.1.

\[
P(U) = k \times P_{\text{max}} + (1 - k)P_{\text{max}} \times U
\]  

(2.1)

where \( P(U) \) is the power consumption of a host when its CPU utilization is \( U \), \( P_{\text{max}} \) is the maximum power of a fully utilized host and \( k \) is the fraction of power consumed by an idle host. It is cost-effective to turn the host off when its utilization is equal to zero. It should be noted that a host still consumes energy when it is turned off. Thus, it should be taken into consideration, for future work, to provide a more accurate energy model off-consumption (i.e., the consumption of plugged-host when it is off). It is observed by [33] that the off-consumption is 15% of the idle consumption. Hence, with the assumption that a server is always turned off whenever it is idle (utilization is equal to zero)
then Eq. 2.2 can model power consumption of the $i$-th host.

$$
P_i(U) = \begin{cases} 
  k \times P_{\text{max}}^i + (1-k)P_{\text{max}}^i \times U_i & U_i > 0 \\
  0.15 \times P_{\text{idle}} & U_i = 0
\end{cases} \quad (2.2)
$$

This power model is originally proposed for a uni-processor, however, most of the common servers use multi-core processors. There are three ways to adapt the mentioned model for a multi-core processor. The most straightforward way is to consider each core as an independent processor which in this case, the power model could be too pessimistic as the cores are located on the same chip, and therefore their power consumption is significantly lower than a multi-processor where processors do not share the same chip. The second way to apply the mentioned model for a multi-core processor is to assume that a multi-core processor is a single processor with a higher computation capacity which consumes more power. In this case, if we assume that the multi-core processor in each server is homogeneous (all processing cores are identical) then the total utilization of the workload assigned to this server can be divided by the number of cores within this server and then we can apply Eq. 2.2 to calculate the power consumption of this server. In fact, it is assumed that the assigned workload to this server is uniformly distributed among its processing cores. Nevertheless, we need a load-balancing algorithm to assign VMs to the cores of a server in a balanced manner. According to this approach, the power consumption of a server is derived by Eq. 2.3 where the server utilization is calculated by the following

$$
U_j(X) = \frac{\sum_{i=1}^{N} u_i X_{ij}}{m} \quad (2.3)
$$

where $m$ and $N$ denote the number of cores and the number of software applications respectively, the set $X$ refers to a particular VM placement, $X_{ij}$ denotes the existence of the $i$-th VM on the $j$-th server. If VM $i$ is placed on server $j$, then $X_{ij} = 1$, otherwise, we have $X_{ij} = 0$. In addition, $U_j(X)$ denotes the utilization of the $j$-th host corresponding to the placement $X$. The third power model considers the load of each processing core and the total power consumption of the multi-core chip is calculated according to the load of each core which is a more sophisticated power model.

**Power Consumption of Network Switches**

The energy consumption of the network is dominated by the switches that form the core of the interconnection networks. The energy consumption of a switch
is related to its hardware type, the number of ports and the port transmission rates [34]. The power model proposed by [35] can be used to calculate the power consumption of a switch. According to [35], the power consumption of a switch is calculated by

\[ P_{\text{switch}} = P_{\text{chassis}} + n_{\text{linecard}} \times P_{\text{linecard}} + \sum_{r=0}^{R} n_{\text{ports},r} \cdot P_{r, U_{\text{port}}} \]  

(2.4)

where \( P_{\text{chassis}} \) denotes the chassis power of a switch which depends on the hardware of the switch, \( n_{\text{linecard}} \) implies the number of linecards plugged into the switch, \( P_{\text{linecard}} \) represents the power consumption of a linecard while no ports turn on. \( R \) denotes the number of configurations for the port line rate (let us assume we have three port line rates including 1 Gbps, 10 Gbps, or 100 Gbps), \( n_{\text{ports},r} \) represents the number of ports working in the rate \( r \), \( P_{r} \) is the power consumption of a port running at rate \( r \), and \( U_{\text{port}} \) corresponds to the utilization of each port. \( P_{\text{chassis}} \) and \( P_{\text{linecard}} \) are static components in the sense that they do not scale with the transmission rate. Thus, even when a switch is totally idle, it still consumes a significant power which is around 70% to 75% of the maximum power consumption of a switch working at the maximum capacity [34]. Accordingly, a promising solution to reduce the energy consumption of the network is to put the idle switches into the sleep mode.

2.3 Industrial IoT

In Industrial IoT (IIoT), a dependable and time-predictable local network is deployed as the backbone of the smart factory to create inter-things (objects and humans) connectivity using an integration of wired and wireless communication networks. The integration of wireless and wired stations constitutes a hybrid network. For wired communications, a set of reliable, secure and time-predictable network mechanisms and protocols are employed such as the HART protocol [36] and Industrial Ethernet. Industrial Ethernet protocols like PROFINET and EtherCAT [37] 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 noises, factory process needs, and harsher environments [38].

According to the fundamental concept of IIoT, a sub-set of objects connect to the system over a wireless network, particularly for sensors, that is referred to as a Wireless Sensor Network (WSN) [39]. For the wireless part of an industrial local network, there also exist reliable, secure and time-predictable
protocols and mechanism in place. WirelessHart [40], ISA100.11a [41], and
IETF 6TiSCH Working Group (WG) [42] are well-known examples of such
industrial wireless communication mechanisms.

To choose a proper networking solution for an IoT device requires to con-
sider several factors such as [43]:

- the scale and size of the IoT network,
- security considerations,
- latency requirements,
- the amount of noisy devices that are affecting the communications links,
- the physical location of the embedded devices,
- the battery size, and
- physical size.

In the following some of the well-known IoT architectures are pointed out.
The Arrowhead framework [44] is one of the IoT architectures proposed to
facilitate collaborative automation by networked devices for five business do-
mains including production (manufacturing, process, and energy), smart build-
ings and infrastructures, electro-mobility, energy production and virtual mar-
kets of energy. It is based on the concept of Service Oriented Architectures
(SOA). The concept of local clouds with well-defined isolation from the open
Internet is used to support some key requirements of automation systems, such
as real-time, security and safety, scalability and engineering simplicity. The
framework can operate with a hierarchical set of core systems allowing a sin-
gle machine to operate its own Arrowhead cloud, allowing local authorization
and orchestration rules. Inter-cloud service discovery is supported meaning
that local clouds can utilize outside services or provide data as a service to
outside consumers [45].

The Internet-of-Everything model is developed by the Architecture Com-
nittee of the IoT World Forum hosted by Cisco. This model includes multiple
levels such as

- physical devices and controllers that control multiple devices,
- connectivity for reliable and timely information transmission between
devices and the network, across networks, and between the network and
low-level information processing level, and
2.3 Industrial IoT

- fog computing that bridges information technology and operational technology.

The Intel IoT platform [46] is another architecture for IoT systems. It is an end-to-end reference model to provide a foundation for seamless and secure connection of devices. The model delivers trusted data to the cloud, and ongoing value through analytics. Intel also delivers a strategy for integrating hardware and software products to support the Intel IoT Platform. Spanning from edge devices out to the cloud, the strategy includes API management and service creation software, edge-to-cloud connectivity and analytics, intelligent gateways, and a full line of scalable processors.

2.3.1 Fog Computing

The strict timing requirement of industrial applications indicates a desire to keep the computation and storage closer to IoT devices. Fog computing is considered as an extension of cloud computing, bringing down the services to the edge of the network, providing guarantees on lower latency [47]. However, all the benefits of the cloud should be preserved with extensions to the fog, including virtualization, orchestration, manageability, and efficiency [48]. Accordingly, fog computing is counted as one of the prominent technologies suggested by IIoT to develop a smart factory. Fog computing is one of the key parts in the Intel IoT platform [46] configured for low latency to provide real-time response. There are multiple different definitions for fog computing in the literature. For example:

- Fog architectures selectively move compute, storage, communication, control, and decision making closer to the network edge where data is being generated in order solve the limitations in current infrastructure to enable mission-critical, data-dense use cases [48].

- Fog computing is a model to complement the cloud for decentralizing the concentration of computing resources in data centers towards users for improving the quality of service and their experience [49].

According to these definitions, in this thesis we assume that a fog node could be any device with a processing capability located in the local area network, excluding the embedded devices that are originating the data. In other words, fog nodes include

- local servers and
• the remaining processing capacity of network devices, such as IoT gateways, mobile base stations, routers, and switches.

2.3.2 Computational Offloading

One of the computing models that is sharing a similar concept with the proposed multi-layer computing model used in IIoT is that of computation offloading [50, 51, 52]. Computation offloading has been developed and used in the context of mobile systems where mobile devices have limited resources, such as battery life, network bandwidth, storage capacity, and processor performance. These restrictions can be dealt with computation offloading: sending heavy computation to resourceful servers and receiving the results from these servers. That is a very similar idea as what is developed in IIoT where, due to the limited processing power and storage capacity of embedded edge devices, a heavy computation is sent for either fog nodes or to the cloud, and the results of the processing is returned to the embedded devices.

2.3.3 Industrial Software Architectures

In this thesis we have looked at existing software architectures used when constructing embedded systems. In particular we have looked at automotive systems that share many properties of the components of a smart factory, e.g., many embedded systems that are interconnected using a fairly sophisticated network architecture. Deployment of software within such an architecture is subject to similar optimization characteristics as deploying software within a smart factory, e.g., partitioning of software will result in different performance, resource usage and timing depending on if software is co-located on a single core or if software will have to communicate over networks.

In the following we will present some key characteristics of one of the well-known software architecture standards to develop component-based applications is AUtomotive Open Software ARchitecture (AUTOSAR) [53].

AUTOSAR

AUTOSAR is widely adopted in automotive and automation industry. As this thesis focus on smart factories, AUTOSAR is a good example when it comes to validating the assumptions and model of the system. Nevertheless, the within this thesis proposed resource efficient method is a general scheme which can
Figure 2.6: AUTOSAR software architecture.

be used for other types of component based embedded real-time applications as well.

An AUTOSAR-based application consists of a set of loosely-coupled Software Components (SWC). Each SWC specifies its input and output ports, and the SWC can only communicate through these ports. All services that are demanded by the SWCs are provided by the AUTOSAR Run-Time Environment (RTE). Indeed, application SWCs are conceptually located on top of the RTE. The RTE is specifically generated for every ECU. The RTE itself uses the AUTOSAR Operating System (OS) and Basic Software (BSW). The RTE is also responsible to map the corresponding system calls to the BSW modules. Accordingly, there is a continuous interaction between SWCs and BSWs. Figure 2.6 depicts this architecture.

The BSW provides a standardized and highly-configurable set of services, such as: communication over various physical interfaces, NVRAM1 access, management of run-time errors. The BSW forms the biggest part of the standardized AUTOSAR environment [54]. The configuration of the basic software on different cores is out of the scope of this thesis, and it remains as one of the concerns of the RTE designers. However, to find an efficient allocation of a SWC onto cores, the location of BSWs communicating with the SWC should be reflected to reduce the communication cost between the SWC and the BSWs.
Chapter 3

Research Summary

3.1 Problem Statement and Research Goals

The overall goal of the thesis is to achieve a resource efficient design of computing and communication resources in a smart factory where timing and other system requirements such as memory, bandwidth limitations and reliability considerations are respected. Various types of resources need to be handled in order to achieve such a goal. As is mentioned in Section 1.1, three research themes are highlighted as the key building-blocks of Industrie 4.0. In the following, we present four corresponding research challenges where the three first challenges are associated with research theme I, II and III respectively, while the last challenge focuses on the system perspective where the given workload need to be divided among the different computing layers of the system.

3.2 Research Challenges

3.2.1 Research Challenge 1 (RC1)

RC1: Optimizing the CPU usage in embedded multi-core processors to run component-based real-time applications

Targeting this challenge we focus on embedded devices hosting a multi-core embedded processor on which a component-based real-time application is run. Although we investigate the efficient use of multi-core processors to run real-time embedded software taking a general perspective, the Automotive Open System Architecture (AU-
TOSAR), as a well-known software standard to develop automotive applications, is taken into account as a case study. An AUTOSAR-based application consists of a set of SWCs. If mapping of SWCs to Operating System (OS) tasks and allocation of tasks to cores are not developed efficiently, a considerable portion of the CPU time is spent to perform the communications. Major factors that are affecting the overall processors’ utilization include

- the inter-SWCs communication cost, and
- the communication cost between SWCs and BSWs.

Being able to minimize the processor utilization can potentially decrease the number of required cores in a multi-core embedded processor while guaranteeing timing constraints.

3.2.2 Research Challenge 2 (RC2)

RC2: Optimizing the number of utilized servers/switches in a cloud data center to minimize energy consumption. Targeting this challenge we focus on a time predictable cloud data center, dubbed as the Real-Time Cloud (RTC), which provides a large pool of computing and storage resources in a time-predictable manner. In other words, the RTC must guarantee the timing requirements of provided services. A resource provisioning mechanism strongly impacts on the consolidation of the given workload on a minimum number of servers and using a minimum number of network switches without any violation of timing constraints. The reduction of the number of utilized servers and switches not only specifies the minimum size of a cloud data center to handle a smart factory but it can also be used to reduce the energy consumption of a data center through putting the idle servers/switches into the sleep mode.

3.2.3 Research Challenge 3 (RC3)

RC3: Optimizing the deployment cost of an IIoT system by reducing the number of required computing and communication resources while timing and reliability constraints are respected. Targeting this challenge we mainly focus on industrial IoT systems as the foundation of a smart factory. IIoT not only provides a communication platform between embedded devices but it is also in charge of collecting the data, processing the data in the level of fog, and eventually sending the data for an external cloud data center. Among
various computing and communication resources existing in IIoT, we particularly focus on

1. sink nodes which are responsible to collect the data from wireless IoT devices and send this data to the cloud via IoT gateways,

2. Software Defined Network (SDN) controllers which are responsible to dynamically alter the network behavior in real-time based on the application requirements, in other words, they adapt the configuration and routing policy of the wireless devices in a smart factory, and

3. local servers which are responsible to fulfill the processing/storage capacity needed by a fog platform to meet application constraints where there is not enough computing/storage capacity on the network devices. In other words, considering the available processing/storage capacity of network devices, the remaining capacity of fog should be provided by a set of local servers.

It should be noted that there are other communication devices in an IIoT system such as routers, relays, gateways, etc. which we do not target regarding minimizing them, in this thesis. We only count on the available computing capacity of them to specify the minimum number of required local servers to build a fog platform.

3.2.4 Research Challenge 4 (RC4)

RC4: Optimizing the deployment cost of a smart factory by efficient allocation of industrial applications among three computing tiers including embedded devices, fog and cloud. Targeting this challenge we look at the vertical allocation of a given workload. In other words, we decide

- which part of a given workload that should be executed locally within embedded devices,
- which part should be executed on fog nodes, and finally,
- which part can be executed in a cloud data center?

The allocation decisions should be made according to

- the timing and other requirements of the industrial applications running in the smart factory, and
- the processing capacity of each computing tier.
3.3 Research Questions

Based on the above-mentioned research challenges and the overall goal, to narrow and focus on the research to be conducted, we now list related research questions, where each research question (Q1-Q4) is associated to research challenges, respectively.

Q1. How can a set of software components of an industrial real-time application be partitioned among multiple processing cores of an embedded multi-core processor to minimize inter-SWCs communication cost, to achieve an efficient CPU usage?

Q2. How can a set of industrial real-time applications be placed on and scheduled in multiple servers of a cloud data center such that a minimum number of servers and network switches are used while respecting timing requirements of real-time applications?

Q3. How many fog nodes including sinks, SDN controllers and local servers are at least required to satisfy timing and reliability requirements of an IIoT?

Q4. How should a given workload be distributed among local resources within the fog and the cloud, such that the deployment cost of the system is minimized and such that all the application requirements are satisfied?

3.4 Research Methodology

The research methodology proposed by [55] is elaborated to cover the goals of this research work. Figure 3.1 depicts different steps of the research methodology besides the transitions between the steps.

We started the research work within a project that we conducted together with our industrial partners (ABB Corporate Research and Volvo Cars). This project was related to the first research theme of this thesis and optimizing the CPU usage of multi-core embedded processors for running automotive applications was the project’s major concern. That is why we started with the given research goal of the first research theme. By studying the literature we identified research challenges and we formulated research questions. Then we, with the completion of this research project, addressed the research questions associated to the first research theme.
3.4 Research Methodology

The two other research themes of this thesis mainly originated from a comprehensive survey on the state of the art given current trends within research of the cloud and IoT research communities, as well as discussions with experts from both academia and industry. Hence, for the two latter research themes, identifying the research goals are fulfilled after a literature review.

For organization of the thesis, we had a more general literature review to find out what is the current state-of-the-art when it comes to resource optimization in a smart factory. We look at the different architectures for industrial IoT systems such that we are able to specify the relation and bridge between the three research themes. We describe research conducted within each research theme with respect to those parts of a smart factory that are highlighted by the corresponding research work. There exist a wide range of computing and communication resources in such a large and complex system (i.e., smart factory). However, due to resource limitations of the research resulting in this thesis, we have decided to focus only on a sub-set of computing and communication resources including processing cores of embedded devices; servers and communication switches of the cloud data center; local servers of the fog platform; and finally sinks and controllers in a smart factory.

Once the research goal(s) are defined, we formulate research challenges. As mentioned, the three first research challenges reflect the horizontal scheduling of the workload on the available resources, to minimize the number of required resources in each research theme. When we completed the research corresponding to the first three research challenges we started to look at the last research challenge to provide a comprehensive resource efficient solution for a smart factory. A sub-set of technical challenges (e.g., set-up of optimization problem etc.) are shared between multiple research challenges, while the rest of them are dedicated to each of the research challenge respectively.

Based on the research challenges, we often study the literature again to formulate the research questions since we need to identify which parts of the challenges that can be addressed by adapting existing solutions, and which parts of them that are open challenges. The next phase is to propose solution(s) for the open problems to deal with the research challenges. Here again, we may need to study major related works with more details to be able to implement and analyze them. For most of the research challenges, more than one solution are presented throughout the overreaching project resulting in this PhD thesis, while only the elaborated version of the work associated to each research theme is reported by this thesis.

The implementation phase mainly relies on the use of simulation. It is worth noting that sometimes during the implementation process we find out
some drawbacks or shortcomings in either the proposed algorithm or the system model. In this case we come back to the proposed model, or algorithm, and we revise it and then resume with the implementation process. Additionally, depending on the results of the evaluation we refine the proposed solution and revisit the later steps.

3.5 Validity of the Results

Running experiments on the data collected from real-world applications is a promising approach that we used for paper A. We used data coming from automotive applications of Volvo. However, due to confidentiality, it was challenging to publish their data and details of their system. Hence, for publishing paper A, we had to use a benchmark provided by Bosch, sharing the same characteristics with the automotive applications that were provided by Volvo. For the other papers (Paper B, C, D, and E) we employed the same data set and benchmarks that have been used by other research works found in the literature.

In all papers that are included in the thesis we have used simulation to evaluate the proposed methods and to compare them with other methods found in the literature. Hence, we need a validity test for our experiments. In the following we discuss validity from a construct, internal and conclusion point of view.

3.5.1 Construct Validity

Construct validity concerns whether the measure behaves like the theory says a measure of that construct should behave or not [56]. If there is an inconsistency in behavior, we need to make sure that either the measure is not correct, or the theory is false.

The construct validity plays an important role in the validation of the simulator environment that we have developed for our research work. In our papers we have investigated the different input parameters using a large set of hands-on experiments and within these experiments we have examined the outputs of the simulator to see if they do follow what is expected by corresponding theory. We have also uploaded an open source version of all the simulators related with this research work, making it available to use and further development by other researchers.
3.5 Validity of the Results

3.5.2 Internal Validity

The internal validity reflects the quality of the data analysis. Indeed, it refers to how well an experiment is done, especially examination that it avoids confounding (more than one possible independent variable cause at the same time). When the chance of confounding is lower in a study, its internal validity is higher. One of the advantages of using simulation is that we can limit the number of parameters that are affecting the output of the system, to more easily capture the impact of a particular parameter(s), whereas, in practice, a higher number of factors may affect the results such that it might be difficult to realize a correct correlation between other parameters and the considered parameter(s). To reflect this concern into our research work in all papers, we not only experiment with the application parameters such as number of tasks, number of VMs, execution time and deadline of tasks, size of data communication between tasks, but we also examine different hardware characteristics such as the number of cores, number of servers, interconnected network connecting processing nodes etc. The effect of changing each of these parameters on the output is investigated separately. This significantly helps us to capture the effect of each of these parameters on the output of the system to reduce the chance of confounding, to achieve an internal validity.

3.5.3 Conclusion Validity

The conclusion validity implies the relation between the treatment and the outcome [56]. The major concern is the ability to state that we got the correct conclusions from the available data or not. This relates to the analysis of the collected data. There are two major threats for the conclusion validity [57]:

- Threat 1: to conclude that there is no relationship when in fact there is. It happens when a relationship is missed or not observed.
- Threat 2: to conclude that there is a relationship when in fact there is not. It happens when things are observed that in reality are not there.

To reflect conclusion validity in our research work, after collecting the data, to conclude the results we do not only rely on a few experiments achieved by changing few parameters, but we also conduct a set of experiments where all the selected parameters are examined and their results have been achieved. Therefore, we do not judge only based on one or a few parameters. The selected parameters are those parameters that we already detect that they can
affect the output, and we intend to investigate their impacts. Of course, there could be a large number of parameters for which we know that they all affect the results, but among those, we go for a subset of parameters that are reflected in our model. For example, we know that if the execution time of a task exceeds the estimated worst case execution time, then it can affect our results but we ignore it for a study where we aim to investigate the effect of inter-task data communication on the system performance. To detect a set of parameters affecting on the results, we have conducted discussions with highly-experienced researchers and engineers from both academic and industry, then the suspected parameters (i.e., those parameters which may have an impact on the results) are specified and examined. We also used the comments from reviewers of the papers, audience of the conferences and the opponent and committee of the Licentiate thesis to find out what parameters are missing in the analysis. For example, for conclusion validity of paper A where we choose an engine management system as a component based embedded system while we aim to make conclusions that are valid for general component-based embedded applications. The major parameters making a difference between other types of such embedded systems are (i) the ratio of computing to communication, and (ii) the number of shared components between transactions (i.e., dependent transactions). For both of these parameters, we have created other benchmarks extracted from the main benchmark where both the ratio of task execution times to communications and the ratio of dependent transactions are changed to explore the behavior of our proposed method for more general component-based embedded applications.

3.6 Discussion Regarding Sensitivity Analysis of Parameters

Here the sensitivity of parameters of the algorithms used in the included papers are discussed. The reason behind the importance of such an analysis is that in four papers included in this thesis (i.e., papers A, B, C, and D of the thesis) we have applied meta-heuristic algorithms where the parameters of the meta-heuristic algorithms significantly impact on the efficiency of the algorithm. However, in Paper E, a greedy heuristic algorithm is proposed, working based on the bin-packing algorithms. The algorithm has a deterministic nature and has no parameter to be tuned.

In all of the papers, excluding Paper E, there are penalty coefficients that are used to integrate the constraints into the goal function. Indeed, all the
3.6 Discussion Regarding Sensitivity Analysis of Parameters

Constraints along with the goal function constitute a single cost function where the target is to minimize the cost function rather than the goal function. The value of penalty coefficients has a great impact on the efficiency of this method. If a coefficient is set to a very small value, then there is the risk of accepting an unfeasible solution as the best solution. On the other hand, if a coefficient is set to a very large value, then the chance of finding a global optimum is reduced.

In Paper A, the effect of the penalty coefficient on the results of the algorithm is illustrated. A similar analysis has been used for determining a proper value for the penalty coefficients of other papers (i.e., Paper B, C and D). For the other parameters of the algorithms such as number of iterations, the number of cases examined in each iteration (i.e., the number of ants in Ant Colony Optimization or the size of population in Imperialist Competitive Algorithm), we use both the state of the art [22, 58] and meta-heuristic handbooks [59, 60] to find out the proper range of these values. After specifying the range of each parameter, we start to check multiple values in each range to determine the proper values of parameters. The point is that the combination of the parameters should be investigated, for example, imagine that we have two scenarios, the first scenario where the number of ants is set to 8 and the number of iterations of the algorithm is 10 may work better than scenario B where the number of ants is set to 12 and the number of iterations is set to 8. Therefore, we have to examine several different combinations to find the most effective values of parameters for which the algorithm outperforms other combinations. In most cases, the number of combinations is enormous. That is the reason for why we have limited the number of cases for each parameter, instead of investigating all the values of the range. Nevertheless, when we observe a meaningful difference between multiple values of a particular parameter, we increase the number of cases for a deeper examination of the parameter. For example, when the range of the number of ants is [4, 20], we may check the result of the algorithm where the number of ants is set to 4, 12, 20. If we found a significant difference between two subsequent cases, we increase the number of cases, e.g., we may run the algorithm for five points including 4, 8, 12, 16, 20.
Identify Research Goals
Identify Research Challenges
Specify Research Themes (Research Theme 1, 2 and 3)
Study Related Works with More Details
Present a Solution
Implement the Solution
Evaluate the Solution
Formulate Research Questions

1. Literature Review
2. Identify Research Goals
3. Start with a Given Research Goal
4. Specify Research Themes (Research Theme 1, 2 and 3)
5. Identify Research Challenges
6. Formulate Research Questions
7. Addressing each Research Question

First Research Theme: 3 ← 1 ← 5 ← 6 ← 7
Second Research Theme: 1 ← 2 ← 5 ← 6 ← 7
Third Research Theme: 1 ← 2 ← 5 ← 6 ← 7
Overall Thesis: 1 ← 2 ← 4 ← 5 ← 6 ← 7

Figure 3.1: Research methodology.
Chapter 4

Contributions and Discussion

In this section, the technical contributions of the thesis are discussed. We start with an overview of the papers included in the thesis, then we indicate the relation between the research papers and research challenges. At the end, my personal contribution in each of the included papers is clearly described.

4.1 Overview of the Papers

The first paper (Paper A¹) in this thesis is targeting resource efficiency in the context of automotive software executing on multi-core processors. The tremendous growth of software features in automotive systems causes the emergence of a trend to use multi-core processors. This trend of using software is resulting in increased software complexity for such systems, and it requires the use of more powerful hardware, such as multi-core processors, to run the software. To manage software complexity and reuse, component-based software engineering is a promising solution. However, there are several challenges inherent in the intersection of resource efficiency and predictability of multi-core processors when it comes to running component-based embedded software. In this paper, we present a software design framework solving these challenges. The framework comprises both mapping of software components onto executable tasks, and the partitioning of the generated task set onto the

processing cores of a multi-core processor. In contrast to previous solutions, this framework allows for dependent transactions of software components where the components have internal states, which is common in embedded and automotive software. The goal of the work presented in this paper is to enhance the resource efficiency of the system by optimizing the software design with respect to 1) the inter-software-components communication cost, 2) the cost of synchronization among dependent transactions of software components, and 3) the interaction of software components with the basic software services resident in the software platform on which the components are executed. The proposed framework is compared with alternative solutions. An engine management system, one of the most complex automotive sub-systems, is considered as a test case, and the experimental results indicate a total CPU utilization reduction of up to 11.2% on a quad-core processor in comparison with the common framework in the literature. Hence, this paper addresses RC1.

In the second paper (Paper B\textsuperscript{2}), cloud computing is receiving an increasing attention when it comes to providing a wide range of cost-effective services. In this context, energy consumption of communication and computing resources contribute to a major portion of the cost of services. On the other hand, growing energy consumption not only results in a higher operational cost, but it also causes negative environmental impacts. A large number of cloud applications, including telecommunication, multimedia, and video gaming, have real-time requirements. A cloud computing system hosting such applications, that requires a strict timing guarantee for its provided services, is denoted a Real-Time Cloud (RTC). Minimizing energy consumption in a RTC is a complicated task as common methods that are used for decreasing energy consumption can potentially lead to timing violations. In this paper, we present an online energy-aware resource provisioning framework to reduce the deadline miss ratio for real-time cloud services. The proposed provisioning framework not only considers the energy consumption of servers but it also takes the energy consumption of the communication network into account, to provide a holistic solution. An extensive range of simulation results, based on real data, show a noticeable improvement regarding energy consumption while keeping the number of timing violations less than 1% in average. Hence, this paper addresses RC2.

In the third paper (Paper C\textsuperscript{3}), based on a pay-as-you-go model, cloud computing provides the possibility of hosting pervasive applications from both academic and business domains. However, data centers hosting cloud applications consume huge amounts of electrical energy, contributing to high operational costs and carbon footprints to the environment. Energy-aware resource provisioning is an effective solution to diminish the energy consumption of cloud data centers. Recently, a growing trend has emerged where cloud technology is used to run periodic real-time applications such as multimedia, telecommunication, video gaming and industrial applications. In order for a real-time application to be able to use cloud services, cloud providers have to be able to provide timing guarantees. In this paper, we introduce an energy-aware resource provisioning mechanism for cloud data centers that are capable of serving real-time periodic tasks following the Software as a Service (SaaS) model. The proposed method is compared against an Energy-Aware version of the RT-OpenStack (EA-RTOpenStack). RT-OpenStack is a recently proposed approach to provide a time-predictable version of OpenStack. The experimental results manifest that our proposed resource provisioning method outperforms EA-RT-OpenStack by 16.01\%, 25.45\%, and 25.45\% in terms of energy consumption, number of used servers, and average utilization of used servers respectively. Moreover, from the scalability perspective, the preference of the proposed method for large-scale data centers is more considerable. Hence, this paper addresses RC2.

In the fourth paper (Paper D\textsuperscript{4}), Industrial Internet of Things (IIoT) is the focus. IIoT is one of the key elements of a smart factory. Software defined networking is a technique that benefits network management in IIoT applications by providing network reconfigurability. In this way, controllers are integrated within the network to advertise routing rules dynamically based on network and link changes. We consider controllers within Wireless Sensor Networks (WSNs) for IIoT applications in such a way to provide reliability and timeliness. Network reliability is addressed for the case of node failure by considering multiple sinks and multiple controllers. Real-time requirements are implicitly applied by limiting the number of hops (maximum path-length) between sensors and sinks/controllers, and by confining the maximum work-


load on each sink/controller. Deployment planning of sinks should ensure that when a sink or controller fails, the network is still connected. In this paper, we target the challenge of placement of multiple sinks and controllers, while ensuring that each sensor node is covered by multiple sinks (k sinks) and multiple controllers (k’ controllers). We evaluate the proposed algorithm using the benchmark GRASP-MSP through extensive experiments, and show that our approach outperforms the benchmark by lowering the total deployment cost by up to 24%. The reduction of the total deployment cost is fulfilled not only as the result of decreasing the number of required sinks and controllers but also selecting cost-effective sinks/controllers among all candidate sinks/controllers. Hence, this paper addresses RC3.

In the fifth paper (Paper E) cloud computing is highlighted as one of the principal building blocks of a smart factory, providing a large 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. Hence, this paper addresses RC3 and RC4.

Mapping of Papers to the Research Challenges

In Table 4.1 the key contributions of each paper are mapped to the research challenges of the thesis. Moreover, Figure 4.1 provides a further clarification for the scope of each paper.

---

4.2 Personal Contribution

The research presented in this thesis is done in collaboration with researchers at ABB Corporate Research, Västerås; Stockholm University, Stockholm; Mälardalen University, Västerås; University of Tehran, Tehran (Iran).

Hamid Reza Faragardi is the main driver of the overall work and also the main author of all included papers. The contribution of the co-authors are briefly described as follows:

- In paper A, both the simulator and the theoretical model were developed by me. The discussions with ABB Corporate Research and my supervisors significantly contributed to polishing of the model and making sure that the assumptions and parameters were closer to real-world automotive and automation systems.

- In the evaluation conducted in Paper B, Aboozar Rajabi who was a previous college of Hamid, currently working as an engineer at University of Le Havre, France, was also involved. However, Hamid has been the main driver of the paper.

- In Paper C, a research team from the University of Tehran was also involved in the development of a simulator for energy consumption of data centers, including Prof. M. Kargahi and Saeid Dehnavi.

- Dr. Hossein Fotohi and Prof. Rahim Rahmani contributed in discussions of Paper D.

- In a collaboration with University of Tehran, we developed a simulation platform for a three-tier computing model including end-devices, fog nodes, and the cloud. This simulator reflects the computing model introduced in Paper E. The prominent features of this simulator comprise (i) end-devices and fog nodes can host multi-core processors, (ii)
for each core of the multi-core processor hosted by a fog node, we can specify the available percentage of processing capacity, e.g., we can say that, in the worst-case, 100% of the core one and 40% of the core two are available and the rest of the processing capacity of the cores is used for the primary task of the fog node, which might be data forwarding and data aggregation. Accordingly, the research team from the University of Tehran, including Prof. M. Kargahi and Saeid Dehnavi, participated in the development of the simulator.

- Prof. Thomas Nolte, Prof. Björn Lisper, Prof. Kristian Sandström, and Dr. Alessandro Vittorio Papadopoulos are supervisors of Hamid and have contributed in discussions and revision of the work.
Figure 4.1: Scope of each paper to address the research challenges.
Chapter 5

Conclusions and Future Work

5.1 Summary

In this thesis we deal with optimizing resource efficiency in the context of a smart factory where

- embedded multi-core devices are central local computing units,
- fog computing is deployed as the major component of the interconnect, following the principles of an industrial IoT, and
- cloud computing is used for global computation.

These are all principal components when it comes to the deployment of a smart factory. For each of these components we have presented an overall research theme with associated research challenges, where optimization is a key solution. Within each research challenge the main concern of optimization is

- the processor utilization, when it comes to the embedded devices,
- the total deployment cost of the system (the number of required local servers, the number of sinks to collect the data from sensors and the number of SDN controllers), when it comes to fog computing, and
the number of utilized servers, to reduce energy consumption in the context of cloud computing.

While targeting all three challenges, we also address real-time aspects inherent in the application requirements, i.e., requirements on the timeliness of software execution.

To not limit ourselves to optimize resource efficiency within a particular component, but to also look at the vertical allocation of workload among multiple computing tiers (i.e., embedded devices, IoT and cloud), we use a similar strategy where

1. the key challenges affecting resource efficiency within each research theme are formulated,
2. the main goal along with the major constraints are modeled as an optimization problem,
3. a heuristic algorithm is proposed to address the optimization problem, and
4. the proposed solution framework is evaluated through a simulation platform where the value ranges of application and system parameters come from real-world benchmarks.

The results imply a clear improvement in terms of the minimum amount of IoT / cloud computing resources required to develop a smart factory. From an end-device perspective, in an embedded multi-core device, the total CPU usage for the considered benchmark is optimized by 11.2%. From a cloud perspective, the number of servers in a cloud data center providing a set of real-time applications for the considered benchmark, is improved by 25.5% in comparison to the state of the art. When the cloud data center network is also taken into account, the total number of active servers and active network switches are noticeably reduced, such that when the inactive resources (i.e., servers and switches) are put into a sleep mode, the energy consumption is reduced by 21% in comparison to a recently proposed method. In terms of network components within the factory used to collect sensor data, our proposed method decreases the total deployment cost of the system by 24%.

Overall, these results all manifest that there is a noticeable potential to optimize the number and amount of resources required to develop a smart factory, while the principal constraints of industrial applications such as timing constraints, reliability, and memory constraints, are respected. These proposed
methods all contribute to create a software tool helping the system designers to find a cost-efficient design of a smart factory.

## 5.2 Future Work

Future work can be conducted in several directions. In the rest of this section, four central directions of future work are highlighted:

- **The first direction concerns the area of the embedded multi-core platform used to run AUTOSAR-based applications. Playing with the basic software components in terms of the number of copies of each of them and their allocation to cores has the potential to improve the efficiency of the system. As future work, the problem of mapping and allocation of application software components can be integrated with the challenges of optimal basic software components configuration. In other words, not only the allocation of application software components to cores should take care of the configuration of the basic software, but also the configuration of basic software can change with regards to the allocation of the application software components to cores to provide a better efficiency.**

- **As the real-time cloud is a new concept there are various directions to continue this research work. The first direction is to deal with how cloud computing can be adopted as an efficient and predictable solution to run automation and automotive applications, where a harder level of timing and availability guarantee is required. One of the interesting solutions for addressing such situations is to interleaving an intermediate computing layer between a factory and a cloud data center to deal with the latency requirements of hard real-time applications. The intermediate layer can be implemented as a private cloud inside of the factory collaborating with a cloud data center. This potentially gives us a more predictability to run real-time applications not only in terms of timing concerns, but also in terms of availability and security of the sensitive data.**

- **Considering the migration of applications from the fog to the cloud and vice versa. It is important to adapt the load of fog nodes with respect to variation of the given workload. When some applications running on fog nodes either are terminated or use a lower amount of resources rather than what we expected before, there is the possibility of migrating an application from the cloud to the fog. In addition, when we under**
estimate the resource demand of an application on fog nodes and it needs a higher amount of resource, we may need to migrate it to the cloud.
Bibliography


Bibliography


