Mälardalen University Licentiate Thesis 330

Dependable Distributed Control System

Redundancy and concurrency defects

Bjarne Johansson

Mälardalen University

Mälardalen University Press Licentiate Theses
No. 330

DEPENDABLE DISTRIBUTED CONTROL SYSTEM

REDUNDANCY AND CONCURRENCY DEFECTS

Bjarne Johansson

2022

School of Innovation, Design and Engineering

ISBN 978-91-7485-567-8
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Acknowledgements

Back in the mid-1980s, when I was seven, I got my first computer, a Commodore 64. Since then, my interest in technology and computers has always been there. Hence, picking up studies towards a Master of Science in computer engineering was an easy choice. A course that led to a journey of embedded real-time system development at ABB, where the thought of post-graduate studies never crossed my mind. One day, there was an information meeting about a research school called ARRAY, hosted by Prof. Thomas Nolte. After that, one thing led to another, and I ended up being one of the participants from ABB. Thank you, ABB, and Thomas, for letting me partake. Prof. Thomas Nolte and Prof. Alessandro Papadopoulos became my academic supervisors, and one could not wish for better supervisors. Even though the schedule is tight, you always make time for valuable discussions, not to mention the positive spins in times of despair. I'm very grateful for all the support. Thank you, Thomas and Alessandro. Mats Rågberger became my industrial mentor from ABB, bringing experience, deep embedded-system knowledge, and constructive industry relevance to the research. Thank you, Mats, for the time, interest, input, and valuable discussions. It has been invaluable and very much apricated. Besides my mentors, I would like to thank all my fellow PhD-students from ARRAY and other constellations for all the discussions and sharing thoughts on post-graduate student life and my colleagues and friends at MDU and ABB for their help and excellent company.

The last sentences go to my family. I want to thank Linnea, my common-law spouse, for all your help and support and the happy reminders that it might be time for other fun activities when I have gotten carried away and the working day is long due. One of the curses of the privilege of working with something that also is an interest - it might be hard to let go. Thank you, Linnea, for your love and encouragement. Finally, I would like to thank my parents, especially my mother Inger, who bought that computer in the 80s. You have always been very positive, loving, and supportive.

Abstract

Intelligent devices, interconnectivity, and information exchange are characteristics often associated with Industry 4.0. A peer-to-peer-oriented architecture with the network as the system center succeeds the traditional controller-centric topology used in today's distributed control systems, improving information exchange in future designs. The network-centric architecture allows for the usage of IT solutions such as cloud, fog, and edge computing in the automation industry. These are IT solutions that rely on virtualization techniques such as virtual machines and containers. Virtualization technology, combined with virtual instance management, provide the famous elasticity that cloud computing offers. Container management systems like Kubernetes can scale the number of containers to match the service demand and re-deploy containers affected by failures.

Distributed control systems constitute the automation infrastructure core in many critical applications and domains. The criticality puts high dependability requirements upon the systems, i.e., dependability is essential. High-quality software and redundancy solutions are examples of traditional ways to increase dependability. Dependability is the common denominator for the challenges addressed in this thesis. Challenges that range from concurrency defect localization with static code analysis to utilization of failure recovery mechanisms provided by container management systems in a control system context.

In this thesis, we evaluate the feasibility of locating concurrency defects in embedded industrial software with static code analysis. Furthermore, we propose a deployment agnostic failure detection and role selection mechanism for controller redundancy in a network-centric context. Finally, we use the container management system Kubernetes to orchestrate a cluster of virtualized controllers. We evaluate the failure recovery properties of the container management system in combination with redundant virtualized controllers - redundant controllers using the proposed failure detection and role selection solution.

Sammanfattning

Sammankoppling och informationsutbyte mellan intelligenta enheter förknippas ofta med Industri 4.0. För styrsystem innebär Industri 4.0 att en plattare arkitektur, där nätverket är centrum för systemet, efterträder dagens controllercentrerade styrsystemtopologi och förbättrar informationsutbytet. Den nätverkscentrerade arkitekturen möjliggör ökad användning av teknik utvecklad för IT-ändamål i styrsystemskontext - teknik som möjliggör en högre grad av flexibilitet i automationssammahang. Flexibiliteten möjliggörs till stor del av virtualisering i form av virtuella maskiner, containers och hanteringen av dessa. Containerhanteringssystem, som Kubernetes, kan öka och minska antalet containers för att matcha behov och omfördela containers som exponerats för fel. I styrsystemsammanhang kan den här typen av system användas för att förbättra tillgänglighet och flexibilitet.

Distribuerade styrsystem är kärnan i automationsinfrastrukturen för kritiska system, tex. driften av oljeplattformar eller kraftverk. I dessa miljöer är hög systempålitlighet och tillförlitlighet fundamentalt. Traditionellt är redundans, tex. duplicering av kritisk hårdvara, tillsammans med högkvalitativ mjukvara ett sätt att tillhandahålla hög tillförlitlighet. Den gemensamma nämnaren för de utmaningar som tas upp i denna avhandling är systemtillförlitlighet. Utmaningarna sträcker sig från lokalisering av parallellexekveringsrelaterade mjukvarufel med statisk kodanalys till containerhanteringssystem.

I avhandlingen utvärderar vi möjligheten att lokalisera parallellexekveringsrelaterade mjukvarufel med hjälp av statisk kodanalys. Vidare presenterar vi en kombinerad mekanism för feldetektering och redundansrollsval för redundanta controllers i en nätverkscentrisk kontext. Slutligen använder vi containerhanteringssystemet Kubernetes för att orkestrera ett kluster av virtualiserade controllrar och utvärdera containerhanteringssystemets feldetektering i kombination med vår mekanism för redundansrollsval och feldetektering.

List of Publications

Papers included in this thesis[1]

Paper A: Bjarne Johansson, Alessandro V. Papadopoulos, and Thomas Nolte *Concurrency defect localization in embedded systems using static code analysis: an evaluation.* In 30th IEEE International Symposium on Software Reliability Engineering (ISSRE), 2019.

Paper B: Bjarne Johansson, Mats Rågberger, Alessandro V. Papadopoulos, and Thomas Nolte. *Heartbeat bully: Failure detection and redundancy role selection for network-centric controller.* In 46th Annual Conference of the IEEE Industrial Electronics Society (IECON), 2020.

Paper C: Bjarne Johansson, Mats Rågberger, Alessandro V. Papadopoulos, and Thomas Nolte. *Kubernetes orchestration of high availability distributed control systems.* In 23rd IEEE International Conference on Industrial Technology (ICIT), 2022.

[1]The included papers have been reformatted to comply with the thesis layout.

Other relevant publications[2]

Bjarne Johansson, Björn Leander, Aida Čaušević, Alessandro V. Papadopoulos, and Thomas Nolte. *Work-In-Progress: Classification of PROFINET I/O configurations utilizing neural networks.* In 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019.

Bjarne Johansson, Mats Rågberger, Alessandro V. Papadopoulos, and Thomas Nolte. *Priority based Ethernet handling in real-time end system with Ethernet controller filtering* In 48th Annual Conference of the IEEE Industrial Electronics Society (IECON), 2022.

[2]Not included in this thesis.

Contents

I

Thesis

Chapter 1

Introduction

Automation and control technology is integral to our everyday lives - integrated into dishwashers, washing machines, and other household aid in our homes. It is also a fundamental part of the critical infrastructure that provides our society with utility services such as power and water. These automation technology solutions are growing in intelligence and utilizing internet connectivity to a higher degree to provide better services. For example, an intelligent refrigerator keeps track of the content's status and gives the owner remote access to that information. Compared to a 1990s refrigerator that only kept cool, the intelligent refrigerator exemplifies the increased interconnectivity and intelligence trend in consumer devices, often referred to as the Internet of Things (IoT) [1]. IoT devices, with various intelligence, connect to other devices, users, and central data servers (the cloud). The IoT trend is enabled by the decreased cost of electronics, offering more computational power per dollar, and the Internet's cost-effective interconnectivity.

The industrial IoT analogy is Industrial Internet or Industrial IoT (IIoT) [2]. IIoT technology lays the ground for what the German government coined as the fourth industrial revolution, Industry 4.0. A revolution signified by increased device intelligence and interconnectivity [3]. Examples of envisioned benefits to reap from Industry 4.0 are lower maintenance costs due to predictive maintenance and highly customizable production chains, to mention a few.

The somewhat conservative process automation industry is also transcending into the era of Industry 4.0, impacting even the core of automation, namely the control system.

Controllers and control systems are systems that control other systems or devices. A Distributed Control System (DCS) is a large-scale control system. A typical DCS consists of multiple interconnected controllers that communicate with each other. A DCS is typically the core of large-scale automation

solutions, for example, the automation solution of a whole plant or factory.

DCSs are the automation foundation in various domains, from offshore oil rigs, harbor crane automation, and paper mills to power generation and water distribution plants. Places where downtime or unplanned events can be costly, have an environmental impact, lead to injuries, or even the loss of human lives. In other words, dependability is fundamental for DCS.

And that is the overall goal of the thesis, addressing and utilizing dependability-related challenges and possibilities imposed by the paradigm shift, further elaborated in the following sections.

1.1 Dependability

Dependability is a comprehensive subject that, a bit simplified, can be said to address a system's trustworthiness—the more critical the domain, the higher the confidence requirements on the system. The term dependability is broad and consists of five more concrete attributes: availability, reliability, integrity, maintainability, and safety [4].

Availability is the proportion of time that the system is available and performing the designated task. As an example, the availability of 99.9% translates to roughly nine hours of downtime per year. Reliability is the probability that the provided service will function in a bounded time interval $[0, t]$. I.e., the likelihood of continuous interrupted service, quantified with a Mean Time Between Failure (MTBF). Integrity is about minimizing the probability of fault-induced improper states in the system. Such as reducing the risk of incorrect system states due to soft errors induced by a bit-flip in a memory cell or on a communication link. Errors can be unintentional, such as cosmic radiation causing a bit-flip, or deliberately created by a malicious intruder's data tampering. Maintainability is the ability to maintain the system, typically including the possibility of replacing hardware and updating software. Safety is the avoidance of severe consequences in case of failure.

1.2 Redundancy

Even though the hardware and software might be of outstanding quality, the probability of failure is never zero - neither for hardware nor software. Redundancy is a common failure measure, dividable into three categories, (i) information redundancy, (ii) temporal redundancy, and (iii) hardware (physical) redundancy [5]. As the name implies, information redundancy is adding redundant information to be able to detect and even correct faults, such as a

checksum to a transferred message. Temporal redundancy typically means repetition, for example, transmitting the same message multiple times to compensate for transient faults in the transmission medium [6]. Hardware duplication is the duplication of hardware, and it can counteract both permanent component failures and temporary disturbance by, for example, duplicating the communication paths [7]. Hardware redundancy is a common failure mitigation strategy and comes in many different flavors—for example, triple redundancy in the aerospace industry [8].

In an industrial automation control system context, hardware redundancy usually comes in two different flavors, one mainly for availability and one primarily for high integrity. Hardware duplication with identical software is the redundancy flavor for increased availability, where a backup controller can resume operation in case of failure of the primary. The primary is controlling the process, and the backup is on standby. The standby level of the backup varies depending on the need; cold, hot, or warm, and relates to the level of backup readiness. Cold means that the backup is not executing at all; a cold standby example is a spare controller that, after manual replacement, resumes the operation of the replaced unit. A backup in hot standby can seamlessly resume operation from the controlled process perspective, while a backup in warm standby can resume operation almost seamlessly. Typically, output signals hold their last value for a known, upper-bound takeover time until the backup has resumed operation. The redundancy pattern described, with one backup ready to take over for one primary, is the one-out-of-two (1oo2) pattern. The 1oo2 pattern is an M-out-of-N (MooN) specialization, meaning that M controllers are active out of a total of N. We call the set N, consisting of the controllers forming the set of redundant controllers, the redundant set.

Parallel execution is another alternative, where all the controllers in the redundant set are active and producing output, and it has the potential to elevate both availability and integrity. If all the controllers in the redundant set produce output, output values are available as long as controllers run. Voting is the process of selecting which controllers' output to use. Parallel execution can also serve as a pure integrity-elevating redundancy pattern with diverse hardware and software. A somewhat simplistic view is to see this as two separate channels based on diverse hardware and software, where one channel produces the data and the other a data checksum, independent of each other. The data consumer typically follows the same two-channel pattern, i.e., two channels computing and comparing the checksum with the received.

1.3 Cloud, fog, and edge

In the wake of Industry 4.0 and Industrial Internet follows more extensive cloud computing use, i.e., the utilization of computational power typically hosted in a data center. Cloud computing is a familiar concept used at a large scale for consumer applications and services, such as social media, web hosting, and streaming. However, cloud computing is not commercialized to any significant extent for the automation industry, especially not for control execution, due to the need for deterministic communication with a known upper bound end-to-end communication time. In addition, some applications require very short communication cycles, which lay in the few millisecond range. The cloud is not the solution for those applications needing quick and precise end-to-end communication times since geographic distance and light speed impose hard limits on the end-to-end communication time.

Along with cloud computing comes fog and edge computing, where fog computing utilizes computational power geographically located between the cloud and the edge of the site network. Edge computing uses computational power on devices at the network's edge, such as cloud connectivity devices. Edge device(s) can also serve as a private cloud, i.e., an exclusive cloud not shared amongst different organizations. The cloud utilizer typically hosts private clouds on on-site local servers such as edge devices. The opposite of private clouds is the data center-hosted public clouds available for all paying cloud customers. Fog and edge computing address the remote cloud communication problem by bringing some of the cloud's computational power and elasticity geographically closer to the controlled process [9, 10, 11].

1.4 Cloud technology and software quality

Clouds in the sky appear when water vaporizes and rises, a twenty-four-seven process without human involvement. Cloud-provided services intend to give a similar carefree existence to their users, i.e., the cloud services are just there and working, much like other utility services, such as power and water. However, behind the service are hordes of hardware managed and utilized with advanced software. Software that provides the cloud with the desired elasticity properties. Behind the software are companies and open-source communities with their employees and contributors. The Cloud-Native Computing Foundation (CNCF) [1] provides an overview of the vast product flora of cloud-related software. Virtualization, especially the more performant container-

[1] https://www.cncf.io/

based lightweight virtualization, combined with an orchestration system such as Kubernetes[2] is the cornerstone in providing the elasticity offered by cloud providers. Elasticity is defined as the capability of scaling the computational power of services to meet the current need [12].

During the early days of computerization, all computer applications ran bare metal on the hardware; then came the operating system, such as DOS and Unix, providing more base services and flexibility, with the flip side of increased software volume and complexity. Today, virtualization and orchestration add additional layers and more software. Software that often executes on multi-core hardware platforms. In other words, the amount of software in society is increasing. The Industry 4.0 revolution with increased intelligence and virtualization is just one example.

1.5 Thesis contribution

As described in the earlier sections, the general trend in society is toward increased use of software-provided services. One example of this is the added "intelligence" and connectivity to traditional functions, like the fridge's cooling function and the washing machine's washing function. Today, they can offer users more services, such as notifications when groceries are missing from the fridge and the laundry is done, or optimize washing powder dosage depending on water properties. In the industry context, the trend of increased interconnectivity and intelligence often goes under the flag of Industry 4.0. This thesis addresses dependability challenges and possibilities brought to the DCS domain by the Industry 4.0 transformation. A transformation that brings more intelligence and flexibility, intelligence and flexibility enabled by software. Hence, we start with a software-related challenge.

A well-defined development process with code reviews and testing is common practice in quality software development. However, bugs with low manifestation probability can slip under the test radar. Concurrency defects are a certain kind of software defects that show low manifestation characteristics. With virtualization, legacy controller software can run in new execution environments – a runtime environment with different traits than the traditional hardware hosting the controller software. A new runtime environment can potentially increase the probability of the manifestation of dormant concurrency defects. Hence, the first contribution of this thesis is to evaluate the possibility of using static code analysis tools to locate concurrency bugs in the source code of embedded real-time systems that provide the core functionality of control

[2] `https://kubernetes.io/`

systems.

Redundancy is another cornerstone when it comes to increasing dependability. The gaining interest in virtualization and network-centric topologies motivated us to research and propose a deployment agnostic failure detection and redundancy role selection solution suitable for time-sensitive control systems.

Finally, we show how orchestrator failure recovery can impact controller availability. We do that in two steps: firstly, by describing the software components needed and the alternatives available to set up a Kubernetes cluster for a containerized controller, and secondly, by measuring failure recovery times. Kubernetes is one of the more well-known container orchestrating systems. Combining the containerized controllers' redundancy mechanism with the orchestrator failure recovery results in a pseudo 1ooN controller redundancy. Where the redundant and virtualized controller utilizes the, in this thesis, presented, failure detection and role selection algorithm. We measure the interupdate time of signals published from the containerized controllers while injecting errors to the container hosting nodes. Measurements provide a basis for understanding the failure recovery of a controller in an orchestrated environment.

Chapter 2

Background and Related Work

Technology constantly evolves, and now and then, evolution takes a more significant step on the evolutionary ladder; a potential paradigm-shifting stride. Some argue that with the growth of technology, paradigm shifts will occur more frequently [13]. Historical examples of technology-induced paradigm shifts are the industrial revolutions. The first industrial revolution introduced machine-aided production and impacted society in the late 18th century. In the 19th century, electrification led to the second industrial revolution, followed by the digitalization enabled third industrial revolution in the 20th century. Today, at the beginning of the 21st century, we are part of another paradigm shift that the German government, in 2014, denoted as the fourth industrial revolution or Industry 4.0. Since then, and even before, the impact of Industry 4.0 and the Industrial Internet has been researched [14, 15, 16]. The work presented in this thesis is motivated by the continuous technology transformation and further highlighted by Industry 4.0 transformations imposed on DCS. It ranges from concurrency defects localization in control system software to redundancy and container orchestrations.

2.1 Software and concurrency defects

As touched upon in earlier sections, controller dependability is essential. The source code of a modern controller firmware can consist of millions of lines of code. Therefore, rigorous development processes with extensive testing are fundamental to controller firmware development. However, defects with low manifestation probability might not result in a detectable failure during the test period, which means the fault can lay dormant and undetected. An example of such a category of defects is concurrency defects.

2.1.1 Concurrency defects

Simplified, one can say that concurrency defects manifest as errors when specific execution interleaving patterns between execution entities occur in an unintended way, resulting in an erroneous application state. However, when scratching further on the surface of concurrency defects, one notices several types of concurrency defects with different characteristics. Therefore, Asadollah et al. [17] propose a classification of concurrency defects based on observable properties.

The inherent interdependence between multiple execution entities can make the manifestation of concurrency defects random. In other words, the manifestation probability can be low and appear non-deterministic. Low manifestation probability makes concurrency defects time-consuming to locate; it can take months [18]. In addition, the push for more intelligent devices, virtualization, elasticity, and interconnectivity driven by Industry 4.0, increase the amount of software used in the automation domain [19, 20, 21, 22]. On top of that, virtualization can potentially change the probability of manifestation of dormant concurrency defects due to changed timing when changing the execution context. For example, when moved to a virtualized environment, legacy software running flawlessly in its original execution context might fail due to the manifestation of dormant concurrency defects exposed only in the new environment.

Concurrency defects mitigation measure can be applied in all phases of development, from modeling [23], code analysis [24] to runtime [25, 26, 27]. In this thesis we focus only on code analysis as concurrency defect mitigation method, further elaborated and motivated in the following section.

2.1.2 Static code analysis

Static Code Analysis (SCA) is the process of checking the source code against a set of predefined rules, rules that often come from a guideline or standard, such as MISRA[1]. Typically, a tool performs the check. SCA tools of today are capable of advanced analysis, including localization of concurrency defects. Examples of such tools are CodeSonar[2] and PolySpace BugFinder[3]. Even though, as mentioned, other methods exist for detecting and thereby mitigating concurrency defects, we choose to focus on SCA. The selection of SCA is motivated by its potential to detect and locate concurrency defects both in

[1] `https://www.misra.org.uk`
[2] `https://www.grammatech.com/codesonar-cc`
[3] `https://www.mathworks.com/products/polyspace-bug-finder.html`

legacy code as well as the latest and greatest developed product feature code, just about to be integrated into a build.

Earlier work has evaluated the SCA tool's capability to detect concurrency defects. Manzoor et al. [28] and Mamun et al. [29] evaluate concurrency defect detection in Java code. When it comes to C and C++, Shiraishi et al. [30] use a C-based test suite, and the Juliet suite [31] is another prominent test suite. However, both have limited C++ code, and C++ and object-oriented design are often used to build the embedded software for DCS, thus motivating a more targeted evaluation. An evaluation using real embedded software and a C++-implemented testbed with common object-oriented patterns. In Paper A we perform such an evaluation using real defects and a C++-based testbed we developed.

2.2 Redundancy and control systems

The controller deployment options increase with the technology evolution and the transition of control systems into the Industry 4.0 era. The deployment alternatives of today are typically limited to controller execution on purpose-built and dedicated hardware. Tomorrow, the options will increase to include controller deployment on shared resources in a virtualized context [21, 32]. Consequently, to not be deployment limiting, functions need to be adapted and made deployment agnostic. One such function is controller redundancy. As mentioned in earlier sections, redundancy is a common way to increase dependability by increasing availability, reliability, and integrity. A redundancy solution based on highly customized hardware is an example of a deployment limiting redundancy realization. The dependency on specialized hardware would make it more challenging to deploy such a solution on generic hardware that does not have customized hardware.

Redundancy as a function is dividable into sub-functions, and the sub-functions required to provide the redundancy function depend on the redundancy pattern. MooN standby redundancy requires synchronizing the state of the primary to the backups. State synchronization allows a backup to resume as the primary without returning to historical value output. I.e., state transfer is needed. A second essential function in a standby redundancy solution is failure detection. A backup must detect that the primary has failed to resume the primary operation. A third key function is role selection, that in a 1oo2 setup is implicit since the failure of the primary means that the only backup should resume the primary role. A MooN setup, where we might have multiple backups, requires role selection to ensure that only one of the existing backups resume in the primary role. Failure detection, role selection, and state transfer

are redundancy functions that no longer should rely on customized hardware realization.

The role selection problem is the leader election problem from the distributed systems domain, applied to a different context and use case - the control system context and the selection of redundancy role use case. As the name implies, the leader election is leader selection amongst a plurality of candidates. Having a leader can be one way to ensure consistency by making the leader responsible for handling shared resources. Similarly, a primary controller is accountable for acting on input values and producing output values. The producers and consumers of input and output values are the control system analogy to the shared resources. The difference lay in the real-time aspects; in case of failure, a new primary must be designated and resume operation within an upper-bound takeover time in the range of 500 milliseconds [21]. The need depends on the application and control system vendors want to keep the guaranteed time low to support the requirements of a broad range of domains.

One of the more well-known leader election algorithms is the Bully algorithm presented by Garcia-Molina [33] in the early 1980s. Since then, there have been many proposals of different leader election algorithms [34], [35], [36], [37], [38] and improvements of the Bully algorithm, for example, the Fast Bully Algorithm (FBA) [39].

Failure detection of a remote process is a well-known problem in the distributed system context. Existing work range from different algorithms, such as [40] and [41], to work that proposes quality of service metrics [42].

Compared to generic distributed systems, redundant control systems have deterministic communication patterns on redundant links, minimizing the probability of communication package loss. Nonetheless, the time sensitiveness in the automation domain dictates that a backup needs to resume the role of a failed primary quickly and with an upper-bound time. Hence, this implies a quick failure detection and role selection. Therefore, a combined failure detection with redundancy role selection targeting a network-centric controller is what we present in Paper B.

2.3 Cloud, fog, and edge

Cloud computing is a growing field and argued to be the long-envisioned computation as a utility service [43, 44, 45]. Capable of providing computational resources on-demand, similar to how other utility services such as water and power are provided and utilized in society today. The reduced local hardware footprint, on-demand scalability, and "pay as you go" are some of the attractive properties cloud computing offers. These properties are desirable from

an automation perspective and have motivated research toward cloud-hosted automation solutions [21]. Research has shown that cloud-hosted automation solutions are possible for soft real-time systems where failure to meet a deadline results in functionality degradation instead of system failure [21, 32].

The control logic applications that the controllers run in an automation solution are often real-time and domain-specific applications with different requirements and tolerance. For example, a building automation solution will likely be a soft real-time solution that tolerates longer and less deterministic end-to-end communication with the I/O, sensors, and actuators interfacing with the real world. Compared to an emergency shutdown function that protects expansive equipment, or even human lives, if a dangerous situation emerges. It is evident that utilizing the cloud, specifically cloud services hosted in a geographically distant location, has challenges. The geographical distance between the controlled process and the cloud impacts communication time and determinism.

Fog computing has emerged as a mitigation to the hard end-to-end communication cycle times imposed by the speed of light and the geographical distance to the data center constituting the cloud [46, 47]. In other words, fog computing is the computational elasticity of cloud computing moved geographically closer to the data consumers/producers to address the problem with indeterministic and possible long communication paths [46, 48]. In [9] Steiner et al. propose that the fog can be the cloud for applications that require fast communication cycles, and the enabler for industrial fog is deterministic communication and virtualization. Pop et al. [10] argue that the Time Sensitive Network (TSN) standard enables reliable and deterministic communication, i.e., that TSN is one industrial fog enabler.

An alternative to fog computing is edge computing, i.e., computation on an edge device. An edge device is typically a somewhat computational powerful device at the edge of the network [11]. The typical use case for edge devices is to process data gathered from devices within the sensor network before sending the data to the cloud. The processing before sending fills two purposes, avoid sending unnecessary data to the cloud and process the data faster, allowing a faster/shorter time to provide a reply back to the process. A cluster of edge nodes can form a computational powerful local and private cloud.

The Open Process Automation™ Standard[4] (O-PAS) defines the Advanced Computing Platform (ACP). ACP is a platform that may host multiple virtual controller functions. A platform that provides scalable physical computing resources, such as CPU cores and memory, to the

[4] https://publications.opengroup.org/p190

virtualized controllers. O-PAS envisions ACP as a private edge-cloud, located on-premises. Therefore, the ACP is suitable for handling computational heavy applications or applications that cannot be adequately distributed onto dedicated controllers.

O-PAS prescribes OPC-UA[5] as the realization of the O-PAS Connectivity Framework (OCF). The virtualized controller in the ACP communicates with the process using OPC-UA. Deterministic and upper-bounded end-to-end communication time is realizable using OPC-UA PubSub combined with TSN. Depending on the requirement, TSN might be optional. OPC-UA and TSN are considered the future of automation communication [19].

2.3.1 Virtualization and elasticity

Virtualization is a cornerstone in the elasticity provided by the cloud. Lightweight OS virtualization, in the form of containers, is a performant alternative to virtual machines [49]. However, virtualization alone is not enough to give elasticity. Here is where the container management system comes into the picture. A container management system matches the available physical resources with the virtualized application requirements. It schedules and deploys containers upon the available resources. In other words, it orchestrates the containers; hence container management systems are often referred to as orchestrators. Rodriguez et al. [50] propose a taxonomy that they use to classify the leading container orchestrators.

The elasticity provided by containers and container management systems bring desirable properties to the DCS context, and earlier research has proposed flexible control architectures based on containers [22, 51]. Another dependability-positive property that container management systems such as Kubernetes[6] provide is failure recovery. If detected, regardless of the reason, software or hardware, a failed container can be redeployed by the container management system, assuming that sufficient resources are available. Vayghan et al. [52] provide a Kubernetes operator that reduces the downtime in case of failure to the range of seconds.

Quick recovery of failed containers could complement or even replace a redundancy solution. To be considered a redundancy replacement, recovery times lower than 500 milliseconds are needed [21]. Container management system failure-recovery can complement a 1oo2 controller redundancy between virtualized controllers and provide a pseudo 1ooN redundancy. Which is what we describe and evaluate in Paper C.

[5] https://opcfoundation.org/
[6] https://kubernetes.io/

Chapter 3

Research Goals

The earlier sections have introduced the transformation of DCS. A shift towards virtualization usage in the DCS domain with the network as the information backbone. Within this change, challenges and possibilities related to dependability arise.

Software quality is fundamental for dependability, where the deployment of existing firmware in virtualized runtimes on new platforms actualizes concurrency defect mitigation interest. The changed execution environment can change dormant concurrency defects manifestation probabilities. Furthermore, the network-oriented and virtualized environment means redundancy solutions that must function in a virtualized context and on dedicated hardware. At the same time, orchestrations provide possibilities for failure recovery with the redeployment of virtualized applications. In other words, the transformation of DCS provides dependability challenges and opportunities. That is the overall goal of the thesis, namely to address the challenge and utilize possibilities related to dependability, spawning from the transformation of DCS. The above and earlier sections provide the introduction and motivation for the concrete goals below.

RG 1: To quantify the SCA tool's capability of concurrency defect localization in complex embedded system software. SCA tools of today have expressed concurrency defect support; however, software constructions and complexity vary greatly. Therefore, quantification based on different constructs and complexity levels is needed to understand the feasibility of using SCA for concurrency defect detection.

RG 2: To find a deployment agnostic approach suitable for electing a primary from a set of multiple backups in a control system context. A typical redundancy deployment in a controller-centric architecture is a 1oo2 setup

with potential specialized point-to-point connections for failure detection, role selection, and state transfer. However, dependency on specialized point-to-point connections limit deployment alternatives; hence more flexible solutions are desirable. A deployment-agnostic solution does not limit the deployment alternatives to specific hardware, and it allows for deployment in a virtualized context.

RG 3: To use container orchestrator properties to increase controller availability. Container orchestrators, such as Kubernetes, have a failure recovery mechanism that could complement traditional redundancy solutions and increase availability.

Chapter 4

Research Process and Methods

The work presented is a result of tight collaboration with industry. Given the author's history in the industry, the research work performed and presented in this thesis is applied research. The aim is to be able to contribute to the landscape intersecting the scientific academia and the engineering industry. We bring industry challenges and experience to the scientific context, striving to produce artifacts that gain both.

The overall method we use in the thesis, summarized in Figure 4.1, is our variant of the hypothetico-deductive method [53]. As mentioned above, it starts with the industrial partner identifying a problem or question that we then form a research challenge around, using the below sequence of questions.

a What is the state-of-the-art in the area of the problem?

b What is the state-of-practice in the area of the problem?

c When a and b is answered, do we have a research challenge? If not, start over.

We address the challenge with a literature search and review. Based on the knowledge gained from the literature found, we form a potential solution or an experiment that addresses the challenge. Lastly, we implement the solution or perform the experiment and evaluate the outcome. If the outcome concludes that we need to revisit an earlier step, we do that. On the other hand, if the knowledge we gained from the literature review answers the challenge, we present the knowledge to our industry partner and start over. The need for re-visiting an earlier step can arise in all steps but is not illustrated in Figure 4.1for simplicity reasons.

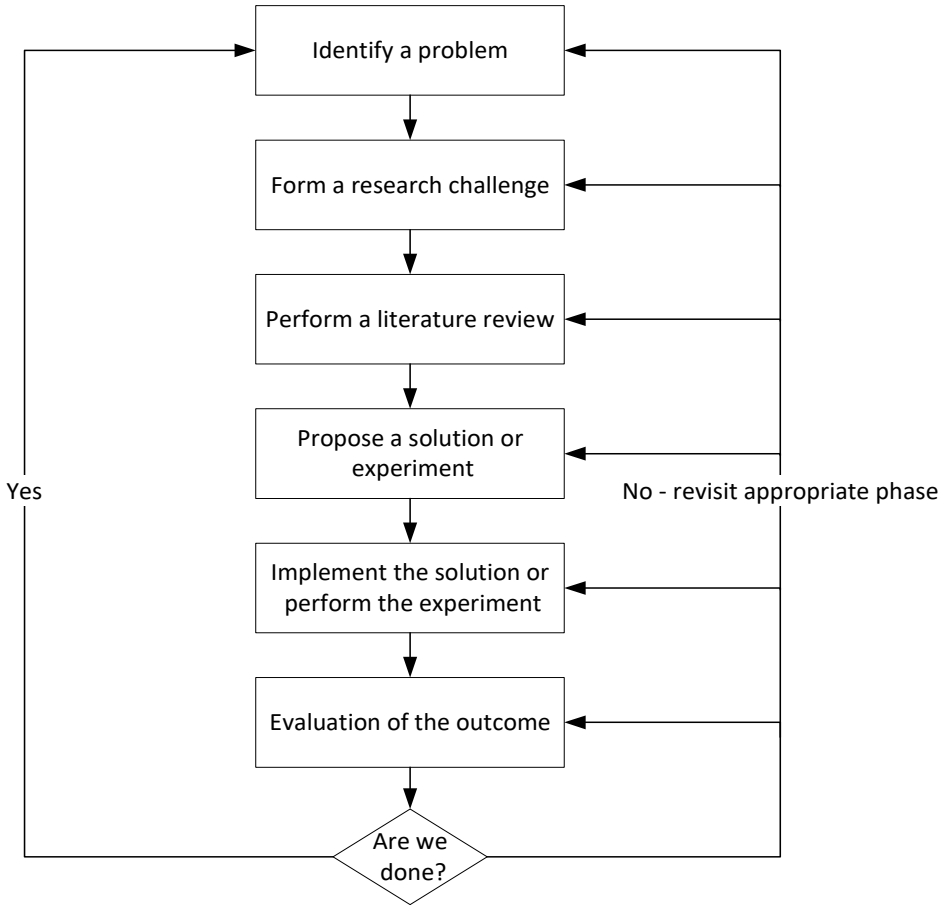

Figure 4.1: The larger research method cycle.

Chapter 5

Thesis Contributions

This chapter presents the thesis contribution and how the contribution connects and relates to the papers included in the thesis. We start with a summary of the contributions and a mapping of the goals. In the final sections, we provide a more elaborated contribution description.

5.1 Contributions

The contributions of the thesis are summarized below. Table 5.1 shows the mapping between the contribution and goals.

- C1: An evaluation and quantification of SCA tools' detection capabilities of concurrency defects in embedded control system software.
- C2: An algorithm for failure detection and redundancy role selection within a redundant controller set. A redundant controller set that can consist of a plurality of backups running on dedicated hardware or virtualized on a shared execution platform.
- C3: A concrete bare-metal Kubernetes cluster architecture for hosting containerized high-availability controllers on embedded devices. Including an overview of the additional software components needed and the configuration alternatives needed for failure recovery.
- C4: Failure recovery time measurements of a single and redundant containerized controller hosted in the Kubernetes cluster. The measures provide the basis for understanding potential orchestrator-induced availability improvements. The orchestrator failure recovery combined with a 1oo2 controller redundancy results in a pseudo-1ooN redundancy.

Table 5.1: Contribution to goal mapping.

	RG1	RG2	RG3
C1	X		
C2		X	
C3			X
C4			X

5.1.1 C1: An evaluation and quantification of SCA tools' detection capabilities of concurrency defects in embedded control system software.

The first contribution of this thesis comes from the work presented in Paper A and is a quantification of SCA tools' capability to detect concurrency defects. Modern controller software can consist of millions of lines of code. The controller is the central part of a DCS system but far from the only one. Communication interfaces providing fieldbus connectivity, I/O modules, actuators, and sensors are examples of other products. Products that also are embedded real-time-system. In other words, the amount of software in a distributed control system is extensive. Typically control system vendors provide and develop these products, which means that the vendor has a large codebase to maintain. The Industry 4.0 transition will likely increase the amount of software even further. Hence, effective software defect detection and mitigation measures are highly relevant. The contribution is the result of the strive towards RG 1. RG 1 originated as a question from our industry partner, whether or not SCA tools are helpful for concurrency defect detection. Since tool-aided SCA is already a part of the development process, the step to add concurrency detection could be small.

Concurrency defects vary in complexity and observable properties [17]. We developed a testbed for the quantification - a testbed that consists of various types of concurrency defects with varying complexity. An example of complexity variation is the function call depth from a thread entry function to a concurrency-wise improperly handled shared resource. Additional to the quantification result, the developed testbed is a sub-contribution. Even though a testbed with deliberately implemented defects can be valuable, it is also advantageous to use actual defects that result from genuine mistakes [54]. Which we also do.

We use the testbed and the real defects to quantify the SCA tool's capability of concurrency defect detection. The result of that work is our first contribution, contribution C1, published and further described in Paper A.

5.1.2 C2: An algorithm for failure detection and redundancy role selection within a redundant controller set. A redundant controller set that can consist of a plurality of backups running on dedicated hardware or virtualized on a shared execution platform.

The result from the quest toward research goal two, RG 2, is the second contribution of this thesis. As stated in RG 2, dependency on specialized hardware reduce deployment alternatives. RG 2 addresses deployment agnostic failure detection and role selection, targeting control system standby redundancy.

As mentioned in earlier sections, redundancy role selection is a variant of the leader election problem. Over the years, the distributed system community has presented many leader election algorithms; one of the more famous is the Bully algorithm [33]. In addition to redundancy role selection, failure detection is essential in a standby redundancy solution. Using a heartbeat is a classic failure detection method; a heartbeat is a cyclic message sent from the supervised to the supervising, a so-called push-based failure detection [55]. Other techniques exist, for example, a pull-based, where the supervisor sends a message to the supervisee and expects a reply.

The second contribution of the thesis consists of an algorithm that combines role selection with heartbeat-based failure detection. An algorithm that we call heartbeat bully. Similar to the original Bully algorithm, the backup with the highest priority wins. The analogy is that it bullies its way to winning by announcing its presence. Heartbeat bully detects failures and elects the replacer within an upper bound selection time. The previous sentence summarizes the essential parts, where the upper bound selection time is crucial for redundant controllers in a control system.

The algorithm is validated using the model checking tool UPPAAL [1] as well as verified in a real system. As a side note, it is worth mentioning that the use of UPPAAL in a real industry use case was also valuable. To show that model checkers, such as UPPAAL, can aid algorithm development in the pre-implementation phase. Furthermore, model checkers are potentially time-savers since the algorithm can be verified before implementation and modified if it does not meet the needs when testing against validation queries.

Contribution two is published and presented in Paper B.

[1] `https://uppaal.org/`

5.1.3 C3: A concrete bare-metal Kubernetes cluster architecture for hosting containerized high-availability controllers on embedded devices. Including an overview of the additional software components needed and the configuration alternatives needed for failure recovery.

Container orchestrators have failure recovery functions, and failure recovery has the potential to increase availability. The third contribution of this thesis comes from addressing research goal three, RG 3. RG3 is about utilizing container orchestrator failure-recovery mechanisms, such as container redeployment of containers hosted on failed compute nodes, in a control system context. We use the O-PAS term Distributed Control Node (DCN) to denote the controller and the term Virtualized DCN (VDCN) for the containerized, virtual controller.

An orchestrator setup is needed to quantify the failure recovery times of virtualized controllers in orchestrated context. Hence we needed to select a container orchestrator. The selection fell on Kubernetes. Kubernetes is one of the most well-known container orchestrator systems. With Kubernetes as a base, we describe the architecture of a Kubernetes-orchestrated cluster of VDCNs.

O-PAS prescribes OPC-UA as the communication foundation. Therefore, the VDCN uses OPC-UA as the communication means. OPC-UA Client-Server for acyclic, request-based communication such as a configuration download from a configuration client and OPC-UA PubSub to exchange cyclic process values to and from, for example, sensors and actuators connected to the physical world. OPC-UA PubSub supports both a broker-based and broker-less mode. Broker-less OPC-UA PubSub utilizes UDP multicast and network infrastructure to decouple publisher and subscriber. The VDCN utilizes broker-less OPC UA PubSub; hence the cluster needs to support UDP multicast. Another consideration for the control domain is that the controllers are typically stateful applications. For example, a VDCN requires its configuration and its internal state. Therefore, the configuration and state must be accessible from all the compute nodes that can host the VDCN. Based on the above-exemplified prerequisites, we describe the architecture and identify suitable components to provide the needed functionality. The results are described and published in paper C and constitute our third contribution.

Table 5.2: Contribution to publication mapping.

	C1	C2	C3	C4
Paper A	X			
Paper B		X		
Paper C			X	X

5.1.4 C4: Failure recovery time measurements of a single and redundant containerized controller hosted in the Kubernetes cluster. The measures provide the basis for understanding potential orchestrator-induced availability improvements. The orchestrator failure recovery combined with a 1oo2 controller redundancy results in a pseudo-1ooN redundancy.

The fourth contribution, like the third, comes from the work towards the third research goal, RG 3. In addition, the second contribution, heartbeat bully, is used as the VDCN redundancy role and failure detection mechanism of the redundant VDCN deployed in the cluster. To understand the usability of Kubernetes failure detection, both as redundancy replacement and as a redundancy complement, we measure the downtime while injecting failure to the compute nodes hosting the single VDCN and the primary VDCN of a redundant VDCN pair. The single configured, i.e., a non-redundant VDCN, is used to measure the recovery times when relying solely on Kubernetes failure detection and recovery. The purpose of the single VDCN is to understand how feasible, based on the quantification, it is to use Kubernetes failure recovery as a redundancy substitute.

The redundant configured VDCN consists of two VDCNs, i.e., two containerized controllers running on two different compute nodes. One is the active primary, the other is the passive backup, a 1oo2 warm standby redundancy, and heartbeat bully provides failure detection and role selection. The VDCN redundancy and Kubernetes failure detection constitute a pseudo 1ooN redundancy. The above summarizes the fourth contribution described and published in paper C.

5.2 Included papers

In this section, the papers included in the thesis are outlined. The papers are mapped to the contributions in Table 5.2.

5.2.1 Paper A

Title: Concurrency defect localization in embedded systems using static code analysis: an evaluation.
Authors: Bjarne Johansson, Alessandro V. Papadopoulos, and Thomas Nolte.
Status: Published at ISSRE 2019.
Abstract: Defects with low manifestation probability, such as concurrency defects, are difficult to find during testing. When such a defect manifests into an error, the low likelihood can make it time-consuming to reproduce the error and find the root cause. Static Code Analysis (SCA) tools have been used in the industry for decades, mostly for compliance checking towards guidelines such as MISRA. Today, these tools are capable of sophisticated data and execution flow analysis. Our work, presented in this paper, evaluates the feasibility of using SCA tools for concurrency defect detection and localization. Earlier research has categorized concurrency defects. We use this categorization and develop an object-oriented C++-based test suite containing defects from each category. Secondly, we use known and real defects in existing products' source code. With these two approaches, we perform the evaluation, using tools from some of the largest commercial actors in the field. Based on our results, we provide a discussion about how to use static code analysis tools for concurrency defect detection in complex embedded real-time systems.
My role: I was the main driver and author of this work, collaborating with my supervisors and industry mentor, who provided valuable input and feedback.

5.2.2 Paper B

Title: Heartbeat bully: Failure detection and redundancy role selection for network-centric controller.
Authors: Bjarne Johansson, Mats Rågberger, Alessandro V. Papadopoulos, and Thomas Nolte.
Status: Published at IECON 2020.
Abstract: High availability and reliability are fundamental for distributed control systems in the automation industry. Redundancy solutions, with duplicated hardware, are the common way to increase availability. With the advent of Industry 4.0, the automation industry is undergoing a paradigm shift; a peer-to-peer mesh-oriented architecture is replacing the traditional hierarchical automation pyramid. With generic computational power provided anywhere in the cloud-device continuum, conventional control-centric solutions are becoming obsolete. The paradigm shift imposes new challenges and possibilities on the redundancy solutions used. We present and evaluate a hardware-agnostic algorithm suitable for failure detection and redundancy role selection in the

new automation paradigm. The algorithm is modeled, evaluated, and validated with the model-checking tool UPPAAL.
My role: I was the main driver and author of this work, collaborating with my supervisors and industry mentor, who provided valuable input and feedback.

5.2.3 Paper C

Title: Kubernetes orchestration of high availability distributed control systems.
Authors: Bjarne Johansson, Mats Rågberger, Alessandro V. Papadopoulos, and Thomas Nolte.
Status: Published at ICIT 2022.
Abstract: Distributed control systems transform with the Industry 4.0 paradigm shift. A mesh-like, network-centric topology replaces the traditional controller-centered architecture, enforcing the interest of cloud-, fog-, and edge-computing, where lightweight container-based virtualization is a cornerstone. Kubernetes is a well-known container management system for container orchestration in cloud computing. It is gaining traction in edge- and fog-computing due to its elasticity and failure recovery properties. Orchestrator failure recovery can complement the manual replacement of a failed controller and, combined with controller redundancy, provide a pseudo-one-out-of-many redundancy. This paper investigates the failure recovery performance obtained from an out-of-the-box Kubernetes installation in a distributed control system scenario. We describe a Kubernetes based virtualized controller architecture and the software needed to set up a bare-metal cluster for control systems. Further, we deploy single and redundant configured containerized controllers based on an OPC UA compatible industry middleware software on the bare-metal cluster. The controllers expose variables with OPC UA PubSub. A script-based daemon introduces node failures, and a verification controller measures the downtime when using Kubernetes with an industry redundancy solution.
My role: I was the main driver and author of this work, collaborating with my supervisors and industry mentor, who provided valuable input and feedback.

5.3 Other publications

Publications listed here are not included in the licentiate thesis:

- Bjarne Johansson, Björn Leander, Aida Causevic, Alessandro Papadopoulos, and Thomas Nolte: *Work-In-Progress: Classification of PROFINET I/O configurations utilizing neural networks*, ETFA 2019.

- Bjarne Johansson, Mats Rågberger, Alessandro V. Papadopoulos, and Thomas Nolte. *Priority based Ethernet handling in real-time end system with Ethernet controller filtering*, IECON 2022.

Chapter 6

Conclusions and Future Work

6.1 Summary of contribution

Overall, we have researched three different goals in this thesis - three goals sprung from a common denominator, the strive towards defying system failure. In other words, we have researched dependability-related topics. The first goal addresses code quality and, more specifically, concurrency defect localization in object-oriented embedded control system source code using SCA. We performed and presented a quantification of the SCA tool's concurrency defect detection capabilities based on actual defects and a developed testbed.

Secondly, we addressed two core functions of a standby redundancy solution: failure detection of the primary and backup selection amongst a possible plurality of candidates. We addressed those core functions in response to the transition of control systems into a more network-oriented era. The result is a failure-detection and redundancy role selection algorithm that we call heartbeat bully.

Lastly, we addressed the utilization of cloud technology, specifically container orchestrators, in a control system context for failure recovery purposes. We used the container orchestrator's failure recovery mechanisms with the redundancy solution of virtualized controllers to provide a pseudo 1ooN redundancy. Additionally, we compared that to the failure recovery of a non-redundant virtualized controller, a non-redundant virtualized controller relying only on the failure recovery provided by the orchestrator.

At first glance, concurrency defects detection using SCA and utilizing container orchestrators, such as Kubernetes, failure recovery might seem like a vastly different field. And undoubtedly, they are. However, they also target the same overall goal. To deliver uninterrupted, properly functioning services to the end-user twenty-four seven. Which, when it all boils down, is what this

thesis is all about.

6.2 Future work

Future work will also address dependability in some form. Even though there are many relevant challenges and potential future work related to concurrency-related problems, our future work will not focus on that. Instead, the focus will lay on network-related challenges motivated by increased network utilization. With the increased utilization of the network comes network functionality convergence. In other words, different functions with different criticality co-exist and share the same underlying network infrastructure.

In shared computing platforms, like the O-PAS proposed ACP, multiple virtualized controllers can co-exist and share the physical resource, including the network resources, where virtual networks can provide an abstraction of the physical network. Virtualized networks and regular network handling require processing. Hence networking requires network resources and computational resources. Nothing new per se; however, the real-time and dependability requirements from the control domain dictate that the systems should be predictable. Hence, the utilization of network resources and network traffic processing must be deterministic when the converged network propagates into the shared and virtualized world. Proper prioritization of network-related processing is relevant for all devices connected to converged networks, such as a network-centric controller.

On top of that, the increased dependency on Ethernet-based networks motivates the investigation of network statistics and information utilization for failure mitigation. For example, even though the network's path between controllers in a redundant set is likely to be duplicated, the risk for islanding/partitioning is never zero. Reducing the probability of undesired behavior with the help of information available from the network is a concrete problem example. Islanding is a well-studied problem in the distributed system community that might need revisiting from a redundant control system context.

The above examples show many challenges related to dependability when control systems and automation find themselves in a new world. Therefore, the future direction of our research will remain under the dependability umbrella but towards the challenges exemplified above.

Bibliography

[1] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information systems frontiers*, 17(2):243–259, 2015.

[2] P Evans and Marco Annunziata. Industrial internet: Pushing the boundaries of minds and machines. *General Electric*, 2012.

[3] R. Drath and A. Horch. Industrie 4.0: Hit or hype? [industry forum]. *IEEE Industrial Electronics Magazine*, 8(2):56–58, June 2014.

[4] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan 2004.

[5] Barry W Johnson. An introduction to the design and analysis of fault-tolerant systems. *Fault-tolerant computer system design*, 1:1–84, 1996.

[6] Inés Álvarez, Ignasi Furió, Julián Proenza, and Manuel Barranco. Design and experimental evaluation of the proactive transmission of replicated frames mechanism over time-sensitive networking. *Sensors*, 21(3):756, 2021.

[7] Rich Hunt and Bogdan Popescu. Comparison of prp and hsr networks for protection and control applications. In *Western Protective Relay Conference, Spokane, WA*, 2015.

[8] Y.C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, 1996.

[9] Wilfried Steiner and Stefan Poledna. Fog computing as enabler for the industrial internet of things. *e & i Elektrotechnik und Informationstechnik*, 133(7):310–314, Nov 2016.

[10] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner. Enabling fog computing for industrial automation through time-sensitive networking (TSN). *IEEE Communications Standards Magazine*, 2(2):55–61, JUNE 2018.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[12] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, June 2013. USENIX Association.

[13] Ray Kurzweil. *The singularity is near: When humans transcend biology*. Penguin, 2005.

[14] L. D. Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014.

[15] J. Li, F. R. Yu, G. Deng, C. Luo, Z. Ming, and Q. Yan. Industrial internet: A survey on the enabling technologies, applications, and challenges. *IEEE Communications Surveys Tutorials*, 19(3):1504–1526, thirdquarter 2017.

[16] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18 – 23, 2015.

[17] Sara Abbaspour Asadollah, Hans Hansson, Daniel Sundmark, and Sigrid Eldh. Towards classification of concurrency bugs based on observable properties. In *COUFLESS '15*, pages 41–47, 2015.

[18] Patrice Godefroid and Nachiappan Nagappan. Concurrency at Microsoft: An exploratory survey. In *CAV ws Expl. Conc. Efficiently and Correctly*, 2008.

[19] Dietmar Bruckner, Marius-Petru Stănică, Richard Blair, Sebastian Schriegel, Stephan Kehrer, Maik Seewald, and Thilo Sauter. An introduction to OPC UA TSN for industrial communication systems. *Proc. IEEE*, 107(6):1121–1131, 2019.

[20] Lucia Lo Bello and Wilfried Steiner. A perspective on IEEE time-sensitive networking for industrial communication and automation systems. *Proc. IEEE*, 107(6):1094–1120, 2019.

[21] T. Hegazy and M. Hefeeda. Industrial automation as a cloud service. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2750–2763, Oct 2015.

[22] Thomas Goldschmidt, Stefan Hauck-Stattelmann, Somayeh Malakuti, and Sten Grüner. Container-based architecture for flexible industrial control applications. *J. Syst. Arch.*, 84, 2018.

[23] M. Shousha, Y. Labiche, and L. C. Briand. A UML/MARTE model analysis method for uncovering scenarios leading to starvation and deadlocks in concurrent systems. *IEEE Tr.Soft.Eng.*, 38:354–374, 2010.

[24] Pär Emanuelsson and Ulf Nilsson. A comparative study of industrial static analysis tools. *E.Notes Theor. Comput. Sci.*, 217:5–21, 2008.

[25] Wenwen Wang, Zhenjiang Wang, Chenggang Wu, Pen-Chung Yew, Xipeng Shen, Xiang Yuan, Jianjun Li, Xiaobing Feng, and Yong Guan. Localization of concurrency bugs using shared memory access pairs. In *29th ACM/IEEE ASE*, pages 611–622, 2014.

[26] Pallavi Joshi, Chang-Seo Park, Koushik Sen, and Mayur Naik. A randomized dynamic program analysis technique for detecting real deadlocks. *SIGPLAN Not.*, 44(6):110–120, 2009.

[27] S. A. Asadollah, D. Sundmark, S. Eldh, and H. Hansson. A runtime verification tool for detecting concurrency bugs in FreeRTOS embedded software. In *17th ISPDC*, pages 172–179, 2018.

[28] N. Manzoor, H. Munir, and M. Moayyed. Comparison of static analysis tools for finding concurrency bugs. In *IEEE 23rd ISSRE Wksp*, 2012.

[29] Md Abdullah Al Mamun, Aklima Khanam, Håkan Grahn, and Robert Feldt. Comparing four static analysis tools for Java concurrency bugs. In *Third Swedish W. on Multi-Core Computing (MCC-10)*, 2010.

[30] S. Shiraishi, V. Mohan, and H. Marimuthu. Test suites for benchmarks of static analysis tools. In *IEEE ISSREW*, pages 12–15, 2015.

[31] Static analysis tool study methodology. Center for Assured Software (CSA), NSA, 2011.

[32] Thomas Goldschmidt, Mahesh Kumar Murugaiah, Christian Sonntag, Bastian Schlich, Sebastian Biallas, and Peter Weber. Cloud-based control: A multi-tenant, horizontally scalable soft-PLC. *2015 IEEE 8th International Conference on Cloud Computing*, pages 909–916, 2015.

[33] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Trans. Comput.*, 31(1):48–59, January 1982.

[34] Md Murshed and Alastair Allen. Enhanced bully algorithm for leader node election in synchronous distributed systems. *Computers*, 1, 06 2012.

[35] M. EffatParvar, N. Yazdani, M. EffatParvar, A. Dadlani, and A. Khonsari. Improved algorithms for leader election in distributed systems. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 2, pages V2–6–V2–10, April 2010.

[36] A. Arghavani, E. Ahmadi, and A. T. Haghighat. Improved bully election algorithm in distributed systems. In *ICIMU 2011 : Proceedings of the 5th international Conference on Information Technology Multimedia*, pages 1–6, Nov 2011.

[37] Minhaj Khan, Neha Agarwal, Saurabh Jaiswal, and Jeeshan Ahmad Khan. An announcer based bully election leader algorithm in distributed environment. In Pushpak Bhattacharyya, Hanumat G. Sastry, Venkatadri Marriboyina, and Rashmi Sharma, editors, *Smart and Innovative Trends in Next Generation Computing Technologies*, pages 664–674, Singapore, 2018. Springer Singapore.

[38] A. Biswas and A. Dutta. A timer based leader election algorithm. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 432–439, July 2016.

[39] Seok-Hyoung Lee and Hoon Choi. The fast bully algorithm: For electing a coordinator process in distributed systems. In Ilyoung Chong, editor, *Information Networking: Wireless Communications Technologies and Network Applications*, pages 609–622, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[40] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Proceedings 2001 Pacific Rim International Symposium on Dependable Computing*, pages 146–153, Dec 2001.

[41] Robbert van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware 1998, pages 55–70, Berlin, Heidelberg, 2009. Springer-Verlag.

[42] Wei Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(1):13–32, Jan 2002.

[43] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[44] D.F. Parkhill. *The Challenge of the Computer Utility*. Number s. 246 in The Challenge of the Computer Utility. Addison-Wesley Publishing Company, 1966.

[45] Taneli Korri. Cloud computing: utility computing over the internet. *Helsinki University of Technology*, 2009.

[46] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Mobile Cloud Comp.*, pages 13–16, 2012.

[47] Shaik Mohammed Salman, Václav Struhár, Alessandro Vittorio Papadopoulos, Moris Behnam, and Thomas Nolte. Fogification of industrial robotic systems: Research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT (Fog-IoT)*, pages 41–45, New York, NY, USA, Apr. 2019. ACM.

[48] Luis M. Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *SIGCOMM Comput. Commun. Rev.*, 44(5):27–32, October 2014.

[49] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *IEEE Int. Symp. Perf. Analysis of Syst. and Software*, pages 171–172, 2015.

[50] Maria A Rodriguez and Rajkumar Buyya. Container-based cluster orchestration systems: A taxonomy and future directions. *Soft., Pract. & Exp.*, 2019.

[51] Václav Struhár, Silviu S. Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro Vittorio Papadopoulos. React: Enabling real-time container orchestration. In *26th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sep. 2021.

[52] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. A kubernetes controller for managing the availability of elastic microservice based stateful applications. *J. Syst. and Soft.*, 175:110924–, 2021.

[53] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*, pages 126–130, 2002.

[54] Shan Lu, Zhenmin Li, Feng Qin, Lin Tan, Pin Zhou, and Yuanyuan Zhou. Bugbench: Benchmarks for evaluating bug detection tools. In *In W. on the Evaluation of Software Defect Detection Tools*, 2005.

[55] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *In ACM Symposium on Applied Computing (SAC 2007*, pages 551–555, 2007.