Service Orchestration for Object Detection on Edge and Cloud in Dependable Industrial Vehicles

Henri Pettinen[1] and David Hästbacka[2,*]

[1]*CrossControl, Tampere, Finland*
[2]*Tampere University, Tampere, Finland*
E-mail: henri.pettinen@gmail.com; david.hastbacka@tuni.fi
[*]*Corresponding Author*

Received 17 November 2020; Accepted 07 May 2021;
Publication 26 August 2021

Abstract

Industrial applications, including autonomous systems and vehicles, rely on processing data on multiple physical devices. The composition of functionality across heterogeneous computing infrastructure is challenging, and will likely get even more challenging in the future as software in vehicles is updated to introduce new features and ensure the safety. New soft real-time use cases emerge and in such cases the model of offloading processing from a limited or malfunctioning device is a viable solution. This study examines orchestration of services across edge and cloud for an industrial vehicle application use case involving image based object detection using machine learning (ML) based models. First, service orchestration requirements are defined taking into account the dependable nature of industrial vehicle applications. Second, an implementation based on Arrowhead framework is presented and evaluated. The open Arrowhead framework offers means for dynamic service discovery, authorization and late binding of computational units. The feasibility of object detection as a service and the suitability of Arrowhead framework to support such orchestrations across edge and cloud is assessed.

Journal of Mobile Multimedia, Vol. 18_1, 1–26.
doi: 10.13052/jmm1550-4646.1811

Keywords: Service-oriented architecture, edge computing, cloud-edge orchestration, arrowhead framework, industrial vehicles, dependable systems, object detection.

1 Introduction

Industrial systems, including autonomous systems and vehicles, are moving towards even more extensive use of data. This includes on the one hand smart and autonomous operation and on the other hand making use of data for various services. Data intensive solutions need to collect massive amounts of data and process it accordingly in order to extract meaningful information for various purposes. Vehicles are evolving towards mobile sensor platforms with great computational power. Most of the operational features are already implemented using electrical components. The role of software running these complex systems is thus increasing. Some of the non-critical functions, e.g. gathering the telematics, can be dispatched to a remote database over a mobile network in a constant interval. In the future, it is expected that some (hard) real-time functionality could be performed outside the vehicle [30, 44] as computational needs arise [27], i.e. close to the edge of the network. This requires an enhanced network infrastructure which is yet to realize on roads or off-road environments. Even with this kind of computational offloading in place, vehicles should implement a fallback functionality themselves, in case networking errors occur.

Cloud computing is an enabling technology for developing software systems and processing large amounts of data in all areas and domains. Cloud computing builds on economies of scale in leveraging performance, scalability and reliability. Edge computing, on the other hand, is characterised by fast processing and short response times possibly in real-time. It has also been characterised as the extension of the cloud in close physical proximity [46] which calls for hybrid (hybrid cloud) solutions combining best of both worlds. All operations, however, do not require millisecond latency nor deterministic networking and there are plenty of applications where offloading the processing close to the edge or to cloud is useful. For example, long term planning or analysis and optimisation of ones own operation.

The composition of functionality across different infrastructures is challenging. In addition to the cloud, on the edge there are typically varying hardware and software platforms used to deploy and execute functionality [16]. New service composition algorithms for handling heterogeneity, handling long-running processes, and tools for managing heterogeneity from

popular programming languages and frameworks need further research as explained by [21]. Furthermore, [46] point out that software frameworks need to implement resilience and methods for failover situations.

One way to achieve highly adaptive system is to embrace the service oriented software architecture (SOA) [41]. The idea is to have one service responsible for one feature. When well designed, a dynamic service replacement can be performed by looking for substitutional services implementing the same functionality. Performing this dynamic service discovery is not a very common practise so far, especially in industrial context.

In this paper, a dynamic service discovery implementing SOA is inspected and evaluated for the need of object detection in the context of dependable industrial vehicles. Research questions and contributions of this paper are as follows: 1. What are the requirements for hybrid cloud orchestration in dependable industrial vehicular applications spanning edge and cloud? 2. How is Arrowhead Framework (AHF) suited for object detection applications in dependable industrial vehicles?

Following, Section 2 presents computing in industrial vehicles, the context of dependability, edge-cloud computing, service-oriented architecture and related works. The AHF framework, used in our implementation and evaluation of the concept, is introduced in Section 3. The requirements for orchestration of services spanning edge and cloud in a dependable context, answering the first research question, are defined and explained in Section 4. The test setup is described in Section 5 and the feasibility of this service orchestration and suitability of AHF to meet the requirements are evaluated in Section 6. A discussion and conclusions are provided in Sections 7 and 8, respectively.

2 Computing in Industrial Vehicles' Context

Gradually happening shift towards new innovation is typical for vehicles. Examples of such transitions are advanced driver assistance systems (ADAS) and over-the-air (OTA) software updates. It is expected, however, that these will become mainstream characteristics of nearly every vehicle [41, 5].

In [30] they envision edge computing to support the more demanding compute tasks that individual vehicles cannot tackle on their own. They claim that the safety and efficient control of e.g. a convoy requires high-frequency exchanges of each vehicle's dynamic data. A roadside edge server and a cloud server can be used to coordinate and manage the vehicles and convoys to go through crossroads safely.

Object detection is one of the most critical and essential abilities of autonomous vehicles. To "see" in different types of environments, multiple different sensors are needed. [22] This explicitly puts higher demand on data throughput and processing capability. Limited processing power and possible real-time demands make this challenging, as studied by [44] in their study on drone computation offloading, claiming that a combination of on-board processing and nearby edge processing can save wireless bandwidth, and thus improve scalability without compromising result accuracy or latency.

Various reasons seem to foster the use of Ethernet-based communication in vehicles. The bandwidth requirements are increasing due to autonomous driving [5]. In addition, Ethernet traffic can be easily transmitted outside the vehicle without major data conversion. Having such a dominant position in various industries and consumer use makes Ethernet a familiar technology for most developers. [15] Real-time communication is also possible using Ethernet and can be complemented also by e.g. industrial cellular networks [43].

2.1 Edge vs Cloud Computing

The term edge computing emerged as the counterforce for cloud computing. When cloud computing centralizes computing resources in remote locations, edge computing is about distributing them close to the data producer. The edge device is typically the first internet connected computer in the data flow pipeline from the data source to the data sink [39, 8].

Edge computing has clear advantages over cloud computing [39, 8]:

- Latency. Round-trip times to edge are presumably shorter than to cloud.
- Privacy. Data travels through fewer processing points and is thus less prone to unauthorized access.
- Resiliency. Fewer intermediate devices routing the signal makes edge computing more resilient in comparison to cloud computing.

Whereas cloud computing has:

- Performance. Cloud computing sets only little physical limits for the computing device.
- Scalability. The performance can be scaled dynamically thanks to the excessive computing infrastructure.
- Accessibility. Edge devices may be behind firewalls or routers providing network address translation. Data flowing to cloud is typically easier to access from anywhere.

2.2 Dependability in Industrial Vehicle Context

A dependable system is a system that is able to provide its intended service with a high probability, and does not cause any harm [33]. Any industrial use vehicle has the potential to cause serious physical harm, economical loss or even loss of life. This sets high requirements for all the systems and subsystems embedded to the vehicles.

Overall dependability is a sum of number of components, including [3]:

- Availability
- Reliability
- Safety
- Maintainability

Continuously a bigger share of the vehicle functionality is done by the Electronic Control Units (ECU) [45]. Consequently, there is a higher chance of faults caused by one of the ECUs or the connections between them [40].

Vehicles are embedded systems, and the software they execute is embedded software. Requirements for embedded software have been, and still typically are, various computing resource limitations (e.g. power efficiency, small memory consumption) and reliability (e.g. fault tolerancy) [26]. Although the embedded computing resources have developed greatly, they still cannot compete against the cloud servers in many areas. Embedded systems in vehicles can be divided in smaller subsystems which are dedicated towards certain applications [23]. Such systems are for instance fuel injection systems or infotainment systems. Some systems have hard real-time requirements, that is, they require a response to an input in specified time constraint. Not meeting the time constraint may result in a catastrophe, or the system may end up in unstable state. [23]

Standards, design principles and software architectures enable complex cyber-physical systems (CPS), such as vehicles, to work. It is essential for the vehicle vendors to make good software architectural decisions and select correct tools in order to maximise their functionality, safety and lifespan.

2.3 Service-oriented Architecture

Service-oriented architecture has been a common implementation strategy for various web applications [4] for the past decade. In industrial vehicles, orchestration of software components at run-time is not common and they are typically bound during design. In SOA, the software system consists of independent and loosely coupled components which provide a single

functionality as a service. There are two kinds of service parties – ones with needs and ones with capabilities to fulfill those needs, typically referred to as consumers and providers. Key requirement for SOA is a mechanism which assists consumers to find the service providers. This can be referred to as service discovery or orchestration. [32]

Benefits of SOA are perhaps not that obvious. Some of the advantages also emerge in the long run. Services as independent and modular software blocks that are reusable, scalable and sharable [32]. The fourth industrial revolution, so called Industry 4.0, comprises ideas of flexible and adaptable automation [25]. SOA suits well for building this kind of industrial applications. Furthermore, these aspects enable new business opportunities by, for example, selling service functionality to other users. Performance wise, scalability is crucial for dynamic load balancing and the application deployment is mitigated by the fact that the application consists of smaller building blocks.

Like all software architectures, SOA has disadvantages. The trade-off of increased flexibility is overhead. The overhead comes in terms of code footprint and messaging. The service discovery process is often the culprit. Moreover, if a system has only one component responsible for the service discovery, the whole system is prone to a single point of failure and stops working if it becomes unavailable.

2.4 Service Interoperability and Automation in Industrial Vehicle Applications

There is a strong need to enhance interoperability and automation among all the new and future software based features and tools used in vehicles [12]. Furthermore, the new features require a high level of security [5, 37].

Various software frameworks are designed to address these needs. According to [34], there are for instance Automotive Open System Architecture (AUTOSAR), BaSys, FIWARE, Industrial Data Space (IDS), Open Connectivity Foundation (OCF) and IoTivity and Open Mobile Alliance (OMA) SpecWorks-LWM2M. Most of them targeted to a rather specific industrial domain.

The AUTOSAR is a software standard mainly targeting the automotive domain, which makes it an interesting option for industrial vehicles as well. AUTOSAR comprises two different platforms, Classic and Adaptive. One of the most notable differences between them is the software architecture. The Classic has a layered architecture whereas Adaptive utilizes service-oriented architecture. This makes Adaptive more suitable for new automotive

applications requiring interoperability and flexibility. Probably the most stringent requirement for AUTOSAR Adaptive compliant applications is that they must be POSIX compliant [1].

Autonomous driving functions may be seen as an application of advanced robotics. Thus it seems natural to have protocols developed on purpose or borrowed from the robotics industry. What once was a purely research related software abstraction for robot development [6], Robot Operating System (ROS), has become mature enough for actual industrial implementations. This is largely due to its new version, ROS2. ROS2 uses Data Distribution Service (DDS) as the communication standard [14]. DDS promotes a decentralized, publish-subscribe communication pattern. ROS2 has official support for Linux, Windows and MacOS [2].

3 Arrowhead Framework as an Enabler of Interoperable Service Composition

To study the orchestration of services across edge and cloud for industrial vehicle applications, we choose to adopt the Arrowhead Framework (AHF). The selection was supported by the fact that it is free and a product of open source collaboration, has an open model for integration, and allows customization [42]. Furthermore, AHF supports security, does not have platform restrictions, and is easily adoptable both on cloud and edge levels [19].

3.1 Framework Overview

AHF is an open service framework [42] defining a generic and distributed model for automating Industrial Internet of Things (IIoT) applications [9]. The framework aims to increase the interoperability between heterogenous industrial systems and devices. In addition, it tries to consolidate the use of SOA in industrial domains, with Industry4.0 based thinking in mind.

AHF is not an IoT platform but a set of rules for implementing a specific SOA philosophy. The rules concern only the communication between the services and systems in different roles and domains. In theory, AHF is programming language agnostic, that is, the communication of all components happens over TCP/IP protocol suite, using HTTP or HTTPS, and the services obey the Representational State Transfer (REST) principles when interacting with the framework. The programming language only needs to provide means for modern web communication.

By utilising the SOA principles, AHF introduces some desirable design for industrial applications. SOA-based distributed functionality allows

loosely coupled services on resource-constrained devices to be interconnected at run-time. For example, a dynamic service discovery can be utilized to switch from faulty service to a functional one. Use of the framework to direct data flows from measurements through dynamic configuration has also been proposed in [20]. Thus, the system introduces increased fault tolerance. In addition, SOA enables distribution of the application components, autonomous interaction between the systems and increases the reusability of software.

3.2 Key Components

AHF comprises a few different abstractions of systems. These are:

- Core Systems – These include the Orchestrator, Authorization and Service registry which provide means for service discovery, authorization and registration, and are mandatory components for every AHF system [42]. A reference implementation of them is provided by the AHF development community.
- Supporting Core Systems – Optional Core Systems that help and extend the functionality of the AHF cloud.
- Application Systems - The service consumers and providers created and controlled by the users. They implement the actual communication logic with the help of the Core Systems and optional Supporting Core Systems. Application systems can expose their services to other application systems.

Figure 1 presents the composition pattern when using AHF, including provider registration, consumer orchestration request, authorization, and

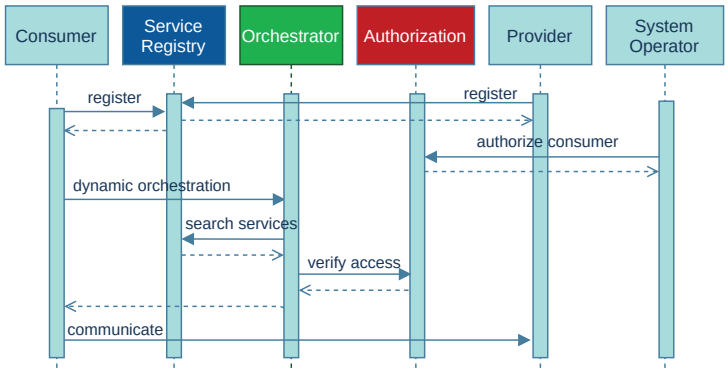

Figure 1 Sequence of dynamic orchestration process using the Arrowhead Framework.

finally consumer-provider communication. It should be noted that AHF is not a middleware for communication but only used initially and all communication between service consumers and providers is direct after the orchestration.

4 Requirements for Dependable Service Orchestration in Hybrid Clouds

The characteristics of industrial, dependable, and autonomous vehicle related computing presented in previous sections are used as the basis when deriving the requirements for hybrid cloud orchestration (service discovery and authorization management). As such we have defined the following:

- High availability – It is unacceptable for dependable systems to experience downtime. Whole SOA becomes inoperative if the orchestration system is unreachable. [24]
- Fast response time – Some dependable applications require fast and almost deterministic response times. If one service provider goes down, the switch to another one needs to happen fast.
- Embedded security and privacy – Security is important for multiple reasons. Dependable systems can become malfunctional if unauthorised systems can access the orchestration system. In addition, user privacy must not be compromised.
- Modifiability – The system must be possible to keep updated. For example, security related methods should be modified if new vulnerabilities are detected. Also new software installations must be supported. [12]

Autonomous driving requires extreme performance and reliability. Let us examine the requirements this sets for hybrid cloud orchestration through examples. When a vehicle is driving and an object blocks the road, it must make fast decisions whether to slow down, stop or dodge the object. If any service that is used to determine the needed maneuver fails to respond, a connection to a new one must be acquired extremely quickly. Another scenario is that an ECU breaks. The services it offers must be then acquired from some other physical device. At the moment, the alternative services with hard real-time requirements must be found in the vehicle.

However, this might not be the case in the future as proposed by [13]. Cloud connectivity can become so ubiquitous and reliable that these services can be deployed off the vehicle. In addition, the connectivity between other vehicles and the roadside infrastructure might become an important enabler

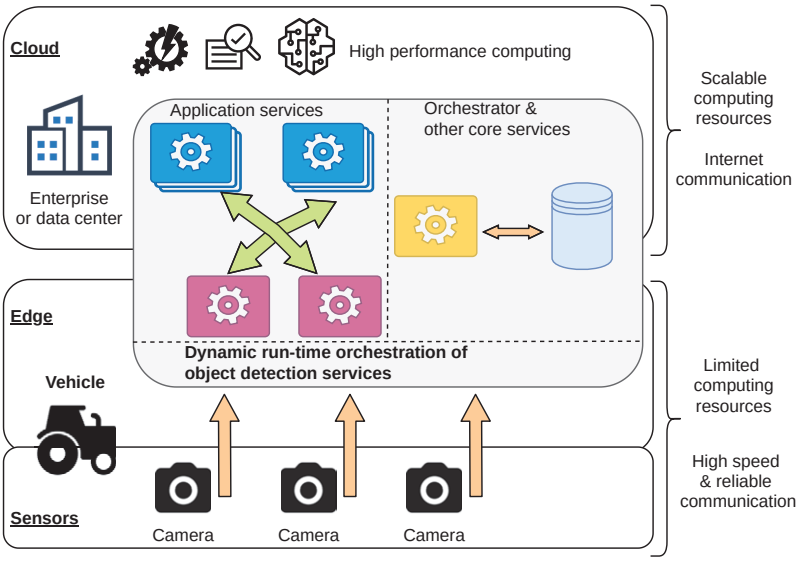

Figure 2 High level overview of a hybrid cloud's SOA for object detection task.

for autonomous driving. Figure 2 distinguishes the different tiers of hybrid cloud computing and their characteristics. Furthermore, it illustrates SOA in dependable hybrid cloud for an object detection use case. As mentioned, object detection is one of the essential functionalities for autonomous driving.

With increased connectivity comes new threats, and a cyber attacker might be able to gain control of the vehicle or disable its functionalities [36]. Thus, security is closely linked to safety and not an optional feature in modern CPS. We see new security exploits constantly happening with servers and other computing devices. Vehicles make no difference when their computing devices are accessible from outside. Hence it is also mandatory to have the possibility to update the software OTA.

Regarding vehicle middleware, Liu [28] lists low overhead and memory footprint, edge–cloud interaction, and security and reliability as the main development areas. Liu et al. [29] examined the suitability of ROS (version 1) for autonomous vehicle platform. They pointed out reliability, performance and security as three shortcomings of ROS in vehicle use. However, they also mention that ROS2 is supposed to solve these issues.

Kugele et al. [24] defined requirements and key quality attributes for automotive SOA. Their listing includes testability which is an important aspect of increasing interest with the growing use of machine learning (ML) based

components. The software architecture should offer feasible and straightforward ways for testing. The importance of this requirement is emphasized in nowadays common continuous integration and deployment (CI/CD) strategy.

The way these requirements need to be addressed from the technical perspective are manifold. First, the system should not introduce a single point of failure. All the components should have a fallback implementation so that the availability is not threatened. In addition, if a service goes down, a new one needs to be searched and provided for the consumer. This procedure includes ensuring that the fallback service is operational. An important concept here is adaptability to situational changes. If the local processing can not be scaled further, the computing tasks need to be dispatched to alternative units. That is, the cloud or other nearby devices.

Second, the system should have adequate performance. The message throughput is an essential characteristic and it would be beneficial if the system could serve multiple requests simultaneously. The system should be implemented using high performance programming languages.

Third, the communication between distributed components must be encrypted and the communicating parties authenticated and authorized. The communication protocol should also strive for efficiency. Fulfilling these requirements should not however introduce a significant delay to the communication.

The system must also be modular in order to support the modifiability requirement. A monolithic application would not work well with resource constrained devices in any case. Modifiablity requires also possibility for remote access. Not all updates can be made by the user or the repair service provider. Even more importantly, new software components must be deployed to vehicles throughout their life-cycle. Thus the system must allow communication with entities outside the vehicle, that is, the local network.

5 Implementing Object Detection as a Service

To address the stated challenges related to vehicle software, and to investigate the feasibility of hybrid cloud computing in vehicles, we introduce a test setup with both edge and cloud processing. In the example, basic object detection tasks are performed for prerecorded video, produced by a frontal view camera of a small industrial machine. The object detection, in our experience, is a novel and interesting topic for mobile machine researchers. Applications linked to object detection also have high dependability requirements which allows us to test the hybrid cloud computing in demanding use cases.

5.1 Object Detection Models

Object detection is a computer vision concept. The object detection task localizes objects in an image and labels these objects as belonging to a target class. [11] In computer vision, a model is a set of processing operations that take inputs, like images or videos, and return pre-learned concepts or labels. An object detection model returns the coordinates and labels of the detected objects.

There are numerous different object detection models and versions of them available. In our implementation, we used two object detection specific models: SSD (Single Shot Multibox Detector) [31] with Mobilenet v2 [38] and YOLO v4 [7] model. These models' internal behaviour is out of the scope of this paper, and is thus not described in detail.

5.2 System Implementation

The main hardware components used in this setup are:

- CCpilot VS display computer (IMX6 board) – the service consumer application on the industrial vehicle
- Raspberry Pi 3B+ – hosting all AHF core services providing service discovery, orchestration and authorization
- Nvidia Jetson Nano – object detection as a vehicle on-board edge processing service
- Nvidia Triton Inference Server[1] – object detection as a remote cloud inference service in a private cloud running on an HP Proliant DL380 server with a Nvidia Tesla P100 GPU.

Figure 3 illustrates how these components are interconnected. On the network's edge are the Raspberry Pi, CCpilot VS display and the Jetson Nano, whilst on the cloud resides the Nvidia Inference Server hosting an Nvidia Tesla P100 GPU. The edge devices are connected through a network switch and the switch is connected to the internet through network routers.

The AHF core services are hosted at the edge on the Raspberry Pi. The used AHF version is 4.1.3.[2] Each device in the setup (Arrowhead cloud) are given a cloud and device specific certificate which is used to both authenticate the device and encrypt the established communication. All the devices and their service offerings are posted to the Service Registry core service. In addition, the consumers are authorised to consume the providing services.

[1] https://github.com/triton-inference-server/server
[2] https://github.com/arrowhead-f/core-java-spring

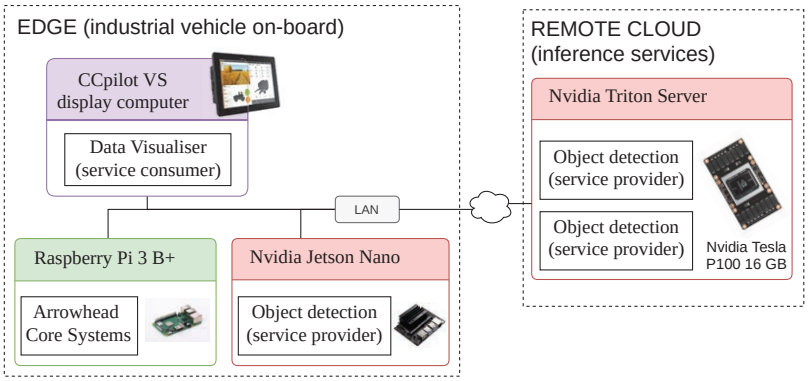

Figure 3 Test setup's key hardware and software components and their interconnection.

All these actions are carried out remotely utilising a cloud specific system operator certificate, as illustrated in Figure 1.

Object detection is implemented as Python applications offering a simple RESTful HTTP(S) service. The services utilise the same Arrowhead cloud specific SSL/TLS certificates to encrypt all the communication and authenticate themselves. The object detection services are deployed both on the edge and in the cloud. The services accept jpeg images as inputs, perform the object detection, and send back the resulting image to the consumer. The CCpilot VS has the role of the service consumer. In the test setup it plays a prerecorded video file, sends frames to object detection service and visualises the output image which is provided in the response on the physical screen. The service provider address is queried from the AHF core services during the startup and whenever the connection is lost to the provider.

5.3 Orchestration for Exchangeable Services

In our setup, the orchestration request is always so called dynamic. In AHF configuration that means that there are no predefined rules to find the appropriate providers for the consumer. Suitable providers are instead searched based on the orchestration request parameters. The dynamic orchestration is set by using so called orchestration flags in the request. Other flags we utilise are the *pingProviders* and *matchmaking*. The former makes the Orchestrator check whether the suitable providers are alive by pinging them. The latter requires the Orchestrator to return only one provider address, which it thinks is the best one.

Both provider systems offer similar remote procedure call (RPC) endpoints for the object detection task. This means that they are fully exchangeable in a plug and play manner. However, their inner implementation is different. First of all, Jetson Nano utilises the SSD Mobilenet v2 model whereas the cloud server has the YOLO v4 model. Before feeding the images to the inference model, both implementations use the OpenCV library for preprocessing. Jetson Nano has its own inference related library called Jetson Inference,[3] which takes care of most of the ML tasks, such as loading the model from memory, communicating with the GPU cores and performing the inference. The cloud implementation utilises Nvidia's Triton server, which hosts the inference model, and thus also performs the inference. Triton is deployed as a prebuilt Docker container which eases the inference setup greatly. The cloud RPC endpoint preprocesses the input images and then sends them in the correct form to Triton. The request is made using gRPC protocol. After the inference is done and the possible objects are identified, both implementations utilise OpenCV again to draw bounding boxes over the objects on the image. Then the images are returned to the consumer.

6 Evaluation and Results

In this section the technical approach from section 5 is tested and evaluated. The suitability of AHF for object detection service orchestration and the feasibility for inference as a service in a dependable industrial vehicle context is evaluated and compared against the derived requirements of Section 4.

6.1 Evaluation Methods

To evaluate the suitability of AHF's service discovery process in the represented test setup we introduce multiple scenarios. By going through these scenarios it is possible to obtain either quantitative or qualitative data. The scenarios are:

1. Disconnection occurs between the object detection provider and consumer application.
2. Load balancing necessary by the orchestrator service (e.g. one provider is too busy).
3. Recovering from brief provider unavailability.

[3] https://github.com/dusty-nv/jetson-inference

Figure 4 Image captured from the video of the industrial vehicle for which object detection using service based inference and service orchestration is studied. On the left the original image and on the right the object detection output with a bounding box.

In the tests we use a prerecorded video filmed from the front of a small wheel loader. On the video, the wheel loader is both stationary and in motion. The frames have size of 1280x720 pixels and the frame rate is 25 per second. The video shows two identifiable objects, a person and a tractor although the object detection itself is not in the scope of this paper. An example excerpt from the video is shown in Figure 4.

Regarding the first scenario, we are particularly interested in how quickly a new service endpoint is obtained. We measure this using a timer in the client application. The timer is set to measure the time between sending the orchestration request and arrival of the response. In Figure 5 the sequence of switching from service provider to another is depicted. After the consumer notices the disconnection or a timeout it sends a new orchestration request to the AHF Orchestrator service. The Orchestrator will internally call Service Registry which will look up for known providers. In addition, it will ping the providers to assure that they are reachable. If there are providers available, the consumer gets a new address to connect to as a response.

The second scenario addresses the situation of multiple consumers overloading one or a few of the providers and in turn some only get a few requests. This can be examined by launching multiple service consumers simultaneously and then inspecting the share of connections to each provider.

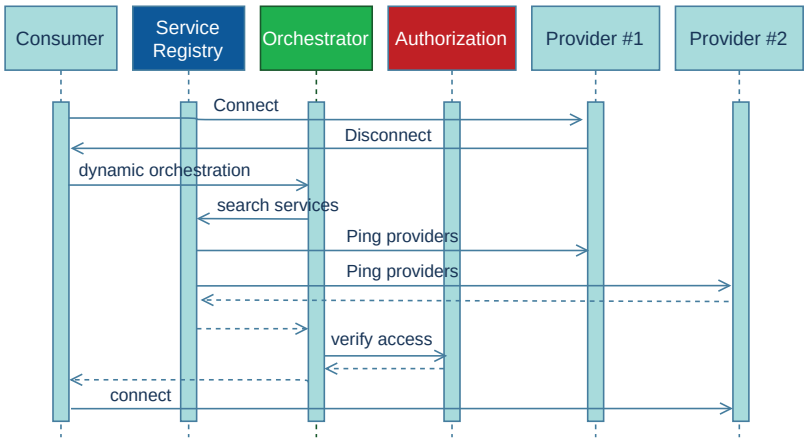

Figure 5 Sequence of switching the service provider using Arrowhead core services.

In the third scenario there is only one service provider available for the consumer. If that goes down briefly, how does the system react? This is simply observed by closing the provider for a while and then restarting it.

6.2 Results from Evaluation Scenarios

The plot in Figure 6 illustrates the response times of 31 orchestration requests for AHF. The average time is 223.8 ms and median is 188 ms. Comparing these to the average times spent for the actual object detection tasks (excluding network delays) gives us perspective. For Jetson Nano, the object detection takes 42.3 ms on average, median being 39.8 (115 samples). For the Nvidia Triton Inference Server with Tesla P100 the same samples took 14.8 ms on average with a standard deviation of 0.366 ms. The underlying models also differ, as explained in Section 5. The inference numbers already give an indication of performance difference but the parallel compute capacity of a powerful GPU versus the Jetson Nano is unrivaled (3584 vs 128 CUDA cores), and the Nvidia Triton Inference Server can serve multiple consumers concurrently.

From the graph it can be seen that orchestration including the authorization process easily takes a significant amount of time that might affect functionality in critical applications. In comparison to object detection operating in near real-time it could mean that the consumer application logic may need to take safety precaution, e.g. slow down or stop its operation, until a service connection is restored.

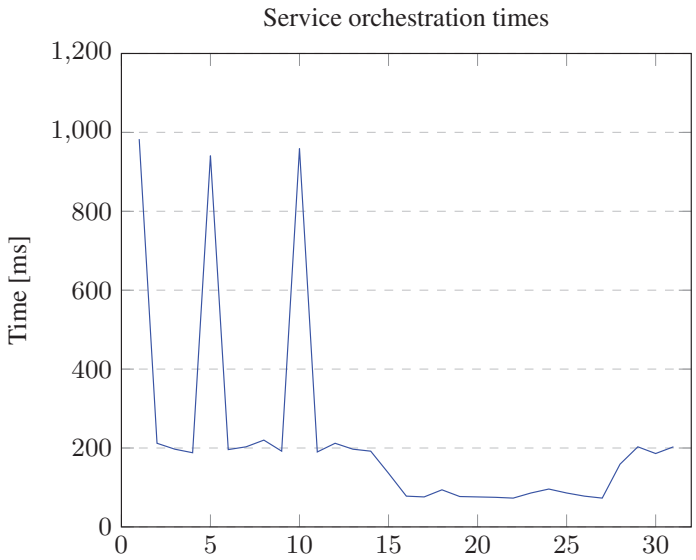

Figure 6 Alternation of service orchestration response times.

The second scenario with load balancing feature is not supported in the AHF community implementation (in the chosen dynamic orchestration mode). However, as the service interfaces between the core services are open it does not prevent one from implementing the orchestrator according to ones needs and thus having load balancing implemented as required for that application. For this study it was not implemented for the test setup.

Going through the third scenario showed that the consumer is able to recover if it starts polling the orchestration when it looses the connection to the provider. The orchestrator will return an empty response immediately if it does not find alive providers. This is avoidable by omitting the *pingProviders* orchestration flag. However, this will most probably lead to the consumer trying to connect to an unavailable service. By pinging all the possible providers, the Orchestrator has the best chance to know the available providers the quickest.

6.3 Evaluating Feasibility of Approach Using AHF

The following requirements were previously defined for dependable service orchestration (see Section 4): high availability, fast response time, embedded security and privacy, and modifiability.

High availability is affected positively to a small extent by the decoupled model of independent service components in the implementation and fundamental nature of AHF. On the negative side, AHF has the core services as a single point of failure and its service model introduces dependency on the network layer for which there are no embedded features for improving or ensuring availability requirements.

Fast and deterministic response time are needed by some applications. As AHF is only brokering connection establishing and not acting as a middleware through which all communication takes place, it enables and allows the implementation of (hard) real-time communication between service consumers and providers. AHF does not mind what protocols the parties use after the orchestration process is completed. Re-orchestration requests, in which AHF takes part, are within reasonable limits as shown in our tests but may require additional safety measures on the consumer side to ensure safe operation in some use cases, e.g. when doing a failover orchestration.

Embedded security and privacy is of increasing importance as systems are connected and multiple systems and devices share the same networks. In the implementation, based on AHF, service consumers and service providers are identified using certificates, and an authorization service grants access tokens in order for consumers to be able to use the provided services. In the implementation, the object detection services implement secure HTTPS interfaces encrypting all requests between each other. As a result the identity of parties, authorization, and integrity of exchanges can be ensured using standard well-established information security practices.

Modifiability is a requirement as CPS and data-driven applications evolve as new use cases emerge. AHF strives to enable dynamic compositions from independent service components and facilitates this requirement. The independent nature means that service providers and service consumers can be developed fully independently, and, due to the loosely coupled service nature, implemented using different implementation methods, tools and platforms, if desired and as shown in our tests. As long as the service interfaces are known and remain unchanged there are no changes needed to the consumers or service providers (neither regarding authorizations or connection strings) as they are always provided by the AHF core infrastructure. In a traditional point to point setup the case is often that all consumers need to be reconfigured independently and, if lacking a trust relationship model, also the service providers need to be reconfigured who is authorized to consume each service. Regarding the remote access requirement, AHF enables it in a secure manner by utilising a special certificate.

7 Discussion

Autonomous vehicles such as cars or machines need to process data from a wide range of sensors in their operation. Additionally, when operating together they will need to process even more information shared from other vehicles. This calls for new methods to distribute intelligence as well as to compose functionality.

An interesting thought is having nodes not restricted by power or computational capacity to serve as additional computing capacity, as suggested by [18], underpinning the need for frameworks and platforms to share the resources. ML is typically compute intensive, especially model training, but increasingly also the inference when multiple video streams are used.

The Nvidia Triton Inference Server providing GPU accelerated inference in combination with a well established service framework shows that inference as a service is a viable solution to implement offloading between restricted devices, the edge and even the cloud, as demonstrated in this study. There can be more and faster compute capacity available from a cloud resource but one needs to take into account the latencies as well.

In the dependable systems context there are still several challenges associated with deterministic networking and computing that are far from solved. Hard real-time networking can be achieved locally but guaranteed compute is still under development unless dedicated hardware solutions are used.

Fortunately there are many use cases where soft real-time is sufficient, e.g. tasks where a higher precision for path planning is obtained from a remote resource whereas immediate actions are computed locally. Another safety implementation strategy is to require additional logic on the control part to ensure safe and proper operation if deterministic communication and computation is not achieved, i.e. slowing down or halting operations until the dependencies are restored to an adequate level.

AHF is an open service framework with several characteristics suited for automating IIoT applications. In comparison to approaches such as OPC UA and DDS it focuses on the service composition layer and authorization models for application ecosystems without interfering in the actual communication between systems. This means that (real-time) communication between individual components can be implemented regardless of AHF. OPC UA has its own supported protocols, including support for Time-Sensitive Networking (TSN), and DDS relies on streaming of data using its real-time publish-subscribe wire protocol. These trade-offs come with limitation in how individual application system components can be implemented. OPC UA interoperability with AHF has been proposed [10] with translator adapters.

Unlike AUTOSAR Adaptive and ROS2, AHF does not specify the message structure between peers. AUTOSAR applications must be made using C/C++ and they need to implement a set of software functionalities to be AUTOSAR compatible, whereas ROS2 and AHF do not limit the use of any programming language. These things considered, it is fair to say that AHF offers a great amount of flexibility in comparison to other similar frameworks. Frameworks such as AHF also target the broader application ecosystems spanning across the Internet. AHF even introduces the concept of inter cloud communication in which AHF instances are relayed to each other using brokers establishing a chain of trust [17]. In the relayed model, however, real-time communication cannot be achieved nor are there any other means available to ensure quality of service among components.

The challenges of authentication and authorization of inter-vehicular signals and services carrying safety commands have been analyzed for vehicle platoons in [35]. In their solution, also utilizing the AHF model, they propose a contract-based approach for specifying safety constraints. Their conclusions also confirm one of our conclusions that this is not always enough and additional safety measures might be needed. An example of this is the need to start a braking sequence before all commands can be authenticated and authorized in the vehicle platoon.

More generally, there is not yet that much edge native similar to what is referred as cloud native on cloud infrastructure. One reason for this is the heterogeneity of edge resources onto which the cloud-edge continuum would need to be extended that would require unified provision of resources, security and a chain of trust of services as well as device hardware, and finally interoperable orchestration and integration models of service components. Such means would allow to seamlessly move computational functions across edge and cloud optimising their trade offs.

8 Conclusion

In this paper we studied orchestration of ML based object detection services across edge and cloud for industrial vehicle applications. First, requirements were defined taking into account the dependable nature of industrial vehicle applications. Following, an implementation based on AHF was presented and evaluated in which object detection services on both edge and cloud were orchestrated to a service consumer application running on an embedded computer typical to industrial vehicles.

As a result it was shown that dynamic orchestration across edge and cloud is possible using AHF and that AHF supports many of the sought-after requirements such as fast response time, security and modifiability. As AHF is not a middleware and doesn't have the means to ensure high availability it is not necessarily adequate for all use cases with deterministic real-time requirements unless client side precautions are taken. It is, however, anticipated that new soft real-time use cases emerge and in such cases the demonstrated model of offloading processing from a limited device is a viable solution and AHF can support that in many ways.

As future work it was identified that services could benefit from a prioritization model, especially when performing inference tasks as a service for various types of needs. If and when these types of service models establish themselves on dependable systems, such as industrial vehicles, more research is also needed on frameworks and service platforms to ensure quality of service both regarding monitoring of individual service executions as well as their compositions.

Acknowledgment

This research has received funding from the EU ECSEL Joint Undertaking under grant agreement no. 737459 and from the partners' national funding authority Business Finland in the Productive4.0 project.

References

[1] Autosar adaptive platform release 19-11 overview. https://www.autosar.org/fileadmin/user_upload/standards/adaptive/19-11/AUTOSAR_TR_AdaptivePlatformReleaseOverview.pdf. Accessed: 2021-03-29.

[2] Ros2 installation. https://index.ros.org/doc/ros2/Installation, October 2020. Accessed: 2021-03-29.

[3] A. Avizienis, J.-C. 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, 2004.

[4] A. Banijamali, P. Heisig, J. Kristan, P. Kuvaja, and M. Oivo. Software architecture design of cloud platforms in automotive domain: An online survey. In *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 168–175, 2019.

[5] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Transactions on Industrial Informatics*, 15(2):1038–1051, 2019.

[6] E. Berger and K. Wyrobek. Stanford personal robotics program. http://personalrobotics.stanford.edu/, May 2008. Accessed: 2021-03-29.

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. https://arxiv.org/abs/2004.10934v1, 2020.

[8] J. Cao, Q. Zhang, and W. Shi. Conclusions. In *Edge Computing: A Primer*, pages 89–90. Springer International Publishing, 2018.

[9] H. Derhamy, J. Eliasson, and J. Delsing. System of system composition based on decentralized service-oriented architecture. *IEEE Systems Journal*, pages 1–12, 2019.

[10] H. Derhamy, J. Rönnholm, J. Delsing, J. Eliasson, and J. van Deventer. Protocol interoperability of opc ua in service oriented architectures. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 44–50, 2017.

[11] K. Fessel. A beginner's guide to object detection. https://www.thisismetis.com/blog/a-beginners-guide-to-object-detection. Accessed: 2020-04-27.

[12] S. Fürst and M. Bechter. Autosar for connected and autonomous vehicles: The autosar adaptive platform. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 215–217, 2016.

[13] M. Gerla, E. Lee, G. Pau, and U. Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 241–246, 2014.

[14] O. M. Group. Data distribution service. https://www.omg.org/spec/DDS/1.4/PDF, March 2015. Accessed: 2021-03-29.

[15] P. Hank, S. Müller, O. Vermesan, and J. Van Den Keybus. Automotive ethernet: In-vehicle networking and smart mobility. In *2013 Design, Automation Test in Europe Conference Exhibition*, pages 1735–1739, March 2013.

[16] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran. The role of edge computing in internet of things. *IEEE Communications Magazine*, 56(11):110–115, November 2018.

[17] C. Hegedus, P. Varga, and A. Frankó. Secure and trusted inter-cloud communications in the arrowhead framework. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 755–760, May 2018.

[18] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen. Vehicular fog computing: A viewpoint of vehicles as the infrastructures. *IEEE Transactions on Vehicular Technology*, 65(6):3860–3873, 2016.

[19] D. Hästbacka, J. Halme, L. Barna, H. Hoikka, H. Pettinen, M. Larranaga, M. Bjorkbom, H. Mesia, A. Jaatinen, and M. Elo. Dynamic edge and cloud service integration for industrial iot and production monitoring applications of industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2021.

[20] D. Hästbacka, J. Halme, M. Larrañaga, R. More, H. Mesiä, M. Björkbom, L. Barna, H. Pettinen, M. Elo, A. Jaatinen, and H. Hoikka. Dynamic and flexible data acquisition and data analytics system software architecture. In *2019 IEEE SENSORS*, pages 1–4, 2019.

[21] A. Huf and F. Siqueira. Composition of heterogeneous web services: A systematic review. *Journal of Network and Computer Applications*, 143:89–110, 2019.

[22] R. Hussain and S. Zeadally. Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys Tutorials*, 21(2):1275–1313, 2019.

[23] H. Kopetz. *Real-Time Systems : Design Principles for Distributed Embedded Applications*. Springer, New York, NY, 2 edition, 2011.

[24] S. Kugele, D. Hettler, and J. Peter. Data-centric communication and containerization for future automotive software architectures. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 65–6509, 2018.

[25] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

[26] E. A. Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, 2008.

[27] X. Li, Y. Dang, M. Aazam, X. Peng, T. Chen, and C. Chen. Energy-efficient computation offloading in vehicular edge cloud computing. *IEEE Access*, 8:37632–37644, 2020.

[28] S. Liu. *Edge Computing for Autonomous Vehicles*, pages 171–181. 03 2020.

[29] S. Liu, L. Li, J. Tang, S. Wu, and J. L. Gaudiot. *Creating Autonomous Vehicle Systems*. Morgan & Claypool, 2017.

[30] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[32] C. MacKenzie, K. Laskey, F. Mccabe, P. Brown, and R. Metz. Oasis standard: Reference model for service oriented architecture 1.0. 08 2006.

[33] P. Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, pages 1–25. Springer International Publishing, 2018.

[34] C. Paniagua and J. Delsing. Industrial frameworks for internet of things: A survey. *IEEE Systems Journal*, pp. 1–11, 05 2020.

[35] R. Passerone, D. Cancila, M. Albano, S. Mouelhi, S. Plosz, E. Jantunen, A. Ryabokon, E. Laarouchi, C. Hegedűs, and P. Varga. A methodology for the design of safety-compliant and secure communication of autonomous vehicles. *IEEE Access*, 7:125022–125037, 2019.

[36] J. Petit and S. E. Shladover. Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):546–556, 2015.

[37] M. Rumez, D. Grimm, R. Kriesten, and E. Sax. An overview of automotive service-oriented architectures and implications for security countermeasures. *IEEE Access*, 8:221852–221870, 2020.

[38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[39] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[40] A. Theissler. Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowledge-Based Systems*, 123:163–173, 2017.

[41] M. Traub, A. Maier, and K. L. Barbehön. Future automotive architecture and the impact of it trends. *IEEE Software*, 34(3):27–32, 2017.

[42] P. Varga, F. Blomstedt, L. L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. M. de Soria. Making system of systems interoperable - the core components of the arrowhead framework. *Journal of Network and Computer Applications*, 81:85–95, 2017.

[43] S. Vitturi, C. Zunino, and T. Sauter. Industrial communication systems and their future challenges: Next-generation ethernet, iiot, and 5g. *Proceedings of the IEEE*, 107(6):944–961, 2019.

[44] J. Wang, Z. Feng, Z. Chen, S. A. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan. Edge-based live video analytics for drones. *IEEE Internet Computing*, 23(4):27–34, 2019.

[45] A. Winning. Number of automotive ECUs continues to rise. https://www.eenewsautomotive.com/news/number-automotive-ecus-continues-rise, 2019. Accessed: 2020-10-27.

[46] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

Biographies

Henri Pettinen received the M.Sc. (Tech.) degree in automation engineering from Tampere University in 2020. His research interests include architectures for cyber-physical systems and edge computing.

David Hästbacka is an Assistant Professor (tenure track) at Tampere University, Finland. He received his D.Sc.(Tech.) degree (with distinction) in 2013 and M.Sc. (Tech.) degree in 2007 at Tampere University of Technology. His research interests are in system and software architectures, and interoperability of software systems in production and energy systems applications.