Towards an On-Demand Redundancy Concept for Autonomous Vehicle Functions using Microservice Architecture

Bo Liu, Victor Pazmino Betancourt, Yimeng Zhu, Jürgen Becker
FZI Research Center for Information Technology
Karlsruhe, Germany
{liu, pazmino, zhu, becker}@fzi.de

***Abstract*—More and more functionalities will be deployed on heterogeneous devices in the vehicle for future autonomous driving. These devices will be connected not only within the vehicle but also to the internet to receive information or consume services provided by other vehicles or road-side units. Even some functions could be offloaded to the cloud infrastructure. However, this high connectivity also means more cyber-security issues for future autonomous driving cars. As cyber-attacks become a more serious issue for the future automotive industry, keeping high availability of safety-critical and non-safety-critical vehicle functions when connected devices in the vehicle are being attacked is an important and challenging task. In this paper, we propose an on-demand redundancy concept to get high availability for autonomous vehicle functions using microservice architecture and container technology. We implemented the concept of embedded devices and showed the feasibility of this concept. The results showed that redundancy could be setup dynamically for non-safety-critical vehicle functions in a cost-effective manner using the proposed approach. This approach could be taken as a security measure while certain devices are being attacked, and the system could continue working without being influenced.**

***Index Terms*—service oriented architectures, autonomous driving, cyber-security, autonomous systems, on-demand redundancy, containerization**

I. Introduction

With the increasing AI applications and growing Car2X communication in the self-driving car, the current E/E architecture is obsolete. It cannot fulfill the requirement of the availability of various functionalities to ensure a safe and comfortable driving experience. [1] For this purpose, more devices in the vehicle will be connected not only with each other but also with the internet to receive information and consume services provided by other vehicles or road-side units. The vehicle's architecture is also changing towards a service-oriented architecture to manage the growing system complexity and provide a more flexible environment. [2] This higher flexibility and connectivity also means a higher possibility of having cyber-security issues. The connections between vehicle and outside world facilitate potential cyber-attacks, making certain functionalities unavailable and affecting the driving experience.

The functionalities in a self-driving vehicle can be categories into safety-critical and non-safety critical where safety-critical functions must satisfy real-time constraints, especially in the automotive domain. Previous studies mainly focus on the fail-operational ability for safety-critical functions with the real-time requirement and propose explicit redundancy of relevant hardware and software to ensure both availability and reliability. [3] However, it is expensive and dispensable to apply this strategy to other vehicle functions. High availability for non-safety-critical functions is also important for the vehicle in the future [3], as it is possible that due to the high connectivity and flexibility of the vehicle architecture, one failure in a non-safety-critical function could lead to a huge problem later.

Concretely, a typical autonomous driving car consists of three modules: perception, planning and controlling module. [4] While perception and controlling modules are usually considered safety-critical, some functionality in the planning module can be viewed as non-safety critical. However, such functionalities still demand high availability in the real application. For example, due to inherently highly dynamic environment in the scenario of autonomous driving, some high-level decision that is usually made in planning module does not endanger the user's life when it fails to work. However, it could generate large costs when it cannot react to a certain event surrounding the vehicle, such as an accident or construction site. Furthermore, with the increasing inter- and intra-vehicle communication boosted by the heterogeneous devices and large scale Internet of Things (IoT) technology, non-safety-critical components should be able to recover from a failure to ensure usability and stability of other partners in the resource-intensive setting. For example, in the distributed path planning setting, the cars are required to move cooperatively to avoid traffic jam. Although the individual planning module's unavailability will not jeopardize the user, the vehicle's cluster can end up in the chaos in such a situation.

In this work, we propose an on-demand redundancy concept using a flexible microservice architecture, which ensures the availability of non-safety components in the self-driving car in a cost-efficient manner. By containerizing the functionality as a service and scheduling the service with container orchestration, the system can migrate the service to other resources on the vehicle when attacking or crashing occurs at runtime. We implement the proposed concept with several edge devices to prove its feasibility and evaluate its performance. The

978-1-7281-8602-3/20/$31.00 ©2020 IEEE

result showed that the on-demand redundancy could be set up dynamically at runtime to get higher function availability and be taken as a security measure against cyber-attacks.

The rest of the paper is organized as follows. Section 2 is the related work of the state of the art research in service-oriented architectures, containerization, and redundancy concept for the future automotive domain. Section 3 presents our proposed concept of on-demand redundancy. Section 4 provides the implementation and evaluation details of our idea. Section 5 is the summary and future work.

II. Background and Related Work

A. Service-oriented Architecture for Automotive

The use of service-oriented architectures (SOA) for automotive is a trending topic in both industry and research because its flexibility fulfills the requirements for future automotive design. Cebotari and Kugele [5] developed a formal model to describe automotive SOA using a classification scheme. Kampmann et al. [6] presented a dynamic SOA concept for highly automated vehicles. Other different methods for designing and modeling SOA for automotive were presented in [7] [8]. Lotz et at. [9] showed the feasibility of changing the software architecture for a driver assistance function to a microservice architecture. The result showed that it could reduce complexity and create a more agile and future-oriented development process if the principles of microservice architectures are followed. The proposed concept in this work also leverages a microservice architecture for its high flexibility and scalability.

B. Containerization for Automotive

There is already some research work on using a containerized environment in the automotive domain for self-driving vehicles. In microservice paper [9], it is already proposed to use the container technology for better modularity and scalability. Wang and Bao [10] adapted a container infrastructure for developing autonomous vehicles. Berger et al. [11] summarized experiences and best practices when using containers and microservices for self-driving vehicles. The feasibility of using data distribution services and containerization for developing automotive software was evaluated in [3]. A similar failover scenario was also considered and tested there. However, the backup service was already defined and deployed on another device, where dynamic service reallocation at runtime is considered and used for the proposed concept in this work.

C. Redundancy Concept for Autonomous Vehicle Functions

A redundancy concept for fail-operational automotive function using dynamic reconfiguration and a model-based development methodology is described in [12] [13]. In order to fulfill real-time requirements of such fail-operational functions, the redundancy function was already deployed on the fallback system, and the communication between the nominal system and the fallback system is done in a service-oriented way. An on-demand redundancy for mapping and scheduling mixed-criticality systems (MCS) is proposed in [14]. Non-safety-critical functions are also considered here. Nevertheless, the focus there is fault-tolerant multi-core MCS, and they did not consider the microservice architecture and container technology in these redundancy concepts.

III. On-Demand Redundancy Concept

The proposed concept consists of two parts. In the first part, a dynamic microservice architecture and its service definition will be introduced, making the concept possible. In the second part, the process of the on-demand redundancy concept is explained based on the well-defined microservice architecture.

A. Dynamic Microservice Architecture

Since the proposed concept is based on the microservice architecture, the services' necessary definitions should be clarified at first. As the core idea of the on-demand redundancy is that the service provider could be changed dynamically during runtime, following dynamic service-related concept should be defined at design time: provided services in the (sub)system, possible service provider, possible service consumer, initial service provider and initial service consumer to make sure that the on-demand redundancy works without a problem at runtime and to have a better overview of possible service behavior.

TABLE I
States Change of Microservice Providers

Device	**Microservice**	
	Service A	***Service B***
Initial Service Provider	Device 1	Device 2
Possible Service Provider	Device 1, Device 2, Device3	Device 2, Device 4
Current Service Provider	Device 1, Device 3	Device 4

The service providers and service consumers are not static anymore with the proposed concept. These could change dynamically according to the changing environment and requirement at runtime. Service could have one to more service providers. Normally, the service is already deployed on these service providers. In our concept, possible providers for one service should be defined without considering if it is already deployed there. This provider is defined as a possible service provider in this work. The service could be deployed on some of the providers in the initial state. The initially selected devices for the deployment of the service is defined as an initial service provider. The deployment could also be changed during runtime. As shown in Table I, the possible service providers for *Service A* is *Device 1*, *Device 2* and *Device 3*. Initially, *Service A* is deployed on *Device 1*. During runtime, *Device 1* have too many tasks and can not fulfill all requirement for *Service A*. As *Device 2* and *Device 3* are also alternative service providers for *Service A*, *Service A* may be dynamically deployed on e.g., *Device 3* after retrieving the deployment from an online or local repository, which contains the service implementation.

Besides the dynamic change of service providers at runtime, another important feature to consider for the proposed concept is the Quality-of-Service (QoS). One service could have different implementations and could be deployed on different devices. Different implementation and different devices could impact QoS, e.g., latency, image quality, etc. That means different service providers could have different QoS-Level for certain services. Service providers with lower QoS-Levels than required could be used as a temporary workaround, but are not suitable to be used as a permanent solution.

B. Process for On-Demand Redundancy Concept

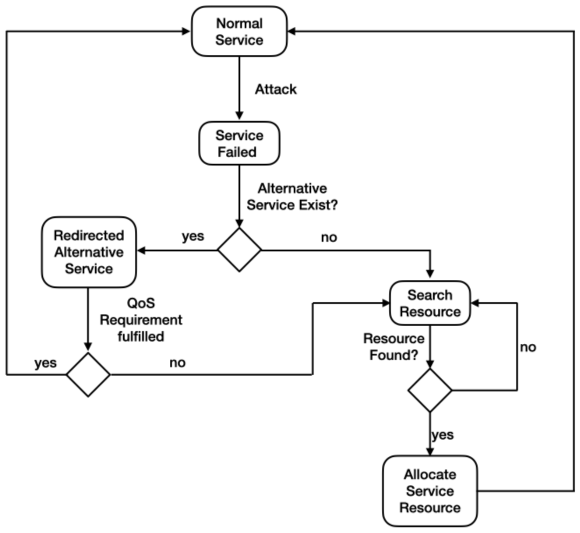
Normal Service
Attack
Service Failed
Alternative Service Exist?
yes
no
Redirected Alternative Service
QoS Requirement fulfilled
yes
no
Search Resource
Resource Found?
no
yes
Allocate Service Resource

Fig. 1. Process for On-Demand Redundancy Concept

After the definition of dynamic service concepts, the basic prerequisites for the proposed redundancy concept are fulfilled. The proposed concept could be used for non-safety-critical vehicle functions to get higher availability and acts as a security measure if certain devices are experiencing a cyber-attack. It can be considered as a self-healing feature. After this dynamic on-demand redundancy comes into play, there could be enough time to analyze the real reason for the function failure: security attack, hardware/software failure, etc. The concrete problem could later be fixed completely with minimal service downtime.

The process of how on-demand redundancy works is shown in Figure 1. There are generally two scenarios for the on-demand redundancy:

1) The service is deployed on only one service provider
2) The service is deployed on several service providers, but some of them have a lower QoS-Level than required

At the initial status, the provided service is working normally. If the current service provider is being attacked and the service becomes unavailable, alternative service providers who have already deployed it will be searched. If it is only deployed one service provider, the service reallocation process should start so that the service will be available after that. If an alternative service provider exists, it will be checked if the QoS Requirement is fully fulfilled. If not, this service provider can only be used as a workaround provider before the service has been reallocated to a service provider with more resources and could fulfill the QoS and other requirements of this service.

IV. Implementation and Evaluation

We implement the aforementioned concept to prove the feasibility and evaluate its performance [1]. Two non-safety critical functions, the object detection for the Heads-up display (HUD) and the speech recognition for voice command, are chosen in our experiment. We simulate the scenario that the object detection function is attacked and shut down, and it can be recovered on other devices later on.

Specifically, we define three microservices in our scenario: the video streaming service, object recognition service, and speech recognition service. These microservices are deployed on a computing cluster with five different devices, as illustrated in Figure 2. With the number marked on the upper left corner, these devices are:

1) A Raspberry Pi 3B with a Google Coral USB Accelerator attached on it (*Coral 1*)
2) A Raspberry Pi 3B (*Pi 3B*)
3) A Google Coral Development Board (*Coral 2*)
4) A Raspberry Pi zero with a Pi Camera attached on it (*Pi Zero*)

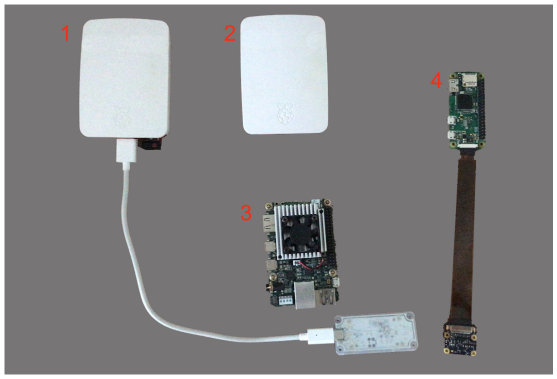

Fig. 2. Hardware Setup

Possible host device for individual service is listed in Table II. Initially, the video streaming service is deployed on Pi Zero, and the object and speech recognition services are deployed on two coral devices. During runtime, the object and speech recognition services can be migrated to coral devices and Raspberry Pi 3B.

Using the terminology in containerization and cloud computing, we define following concepts in our cluster configuration:

[1]The source code can be found in https://github.com/YimengZhu/On-Demand-Redundancy

TABLE II
INITIAL STATE OF MICROSERVICES

Device	**Microservice**		
	Video Streaming	***Object Recognition***	***Speech Recognition***
Service Consumer	Coral 1	Pi 3B	Pi 3B
Initial Service Provider	Pi Zero	Coral 1	Coral 2
Possible Service Provider	Pi Zero	Coral 1, Coral 2, Raspberry Pi 3B	Coral 1, Coral 2, Raspberry Pi 3B

We containerize the object and speech recognition in *Docker* [2], and use *K3s* [3] and *swarm*[4] for container orchestration. The software architecture is shown as in Figure 3. As shown in the figure, docker engine is deployed on *Coral 1, 2*. The containerized object and speech recognition application are deployed on *Coral 1* and *Coral 2* respectively. *Pi 3B* acts as the k3s master which is responsible for the container orchestration. A workaround version of object recognition is also deployed on it. *Coral 1* and *Coral 2* are defined as k3s client. The video streaming service is implemented with Motion [5] on *Pi Zero*. All these devices are connected in the same network. *Coral 1, 2* and *Pi 3B* are declared in the same cluster with k3s configuration.

- **Node** is the working machine on which the service applications can be running, which contains:
  - Node 1: *Coral 1*
  - Node 2: *Coral 2*
  - Node 3: *Pi 3B*
- **Pod** is a running instance of a certain service in the clusters. There are two types of pods in our cluster:
  - Speech Service Pod, in which the speech recognition application is running.
  - Visual Service Pod, in which the object recognition application is running.
- **Deployment** is a group of identical Pods of a certain service, where the number of pods is specified by the *replica* parameter. In our cluster, both speech and visual service have the replica of one.

Using this configuration, the dynamic reallocation for the object recognition service can be realized. If *Coral 1* is attacked, the service will be reallocated to *Coral 2* so that the service stays available.

We evaluated two scenarios as describe in Fig 1:

1) The service is deployed on only one service provider
2) The service is deployed on several service providers, but some of them have a lower QoS-Level than required

[2]https://www.docker.com
[3]https://k3s.io
[4]https://docs.docker.com/engine/swarm/
[5]https://motion-project.github.io/index.html

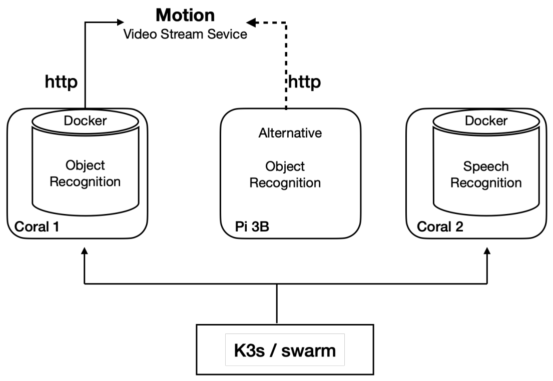

Fig. 3. Software Setup

For scenario 1, the service to be reallocated is the object recognition service, which is initially deployed on *Coral 1*. The possible service providers are *Coral 2* and *Pi 3B*. After *Coral 1* failed, it takes some time for the K3s master to detect the failure and it has to wait for some time to confirm that the service provided by *Coral 1* was not available any more. Then *Coral 2* was scheduled as the new worker node to provide the service and the reallocation process could be triggered. As the service was not deployed on the newly scheduled node, the reallocation process include three part: pull image, create container and start container.

For scenario 2, we decided to deploy the service on *Pi 3B*. The *Pi 3B* has a lower performance for the object recognition service than *Coral 1* and *Coral 2*, as it is a machine learning based function which runs faster on AI accelerators (e.g. Coral). The general process is similar to scenario 1. The reallocation process could be omitted, because the image and container already existed on the device so that the container could start right after the confirmation of failed node (*Coral 1*). As the reallocated service has a lower performance, so it could only be used as a temporary solution. At the end, the service is redeployed on *Coral 2* as a fully functional redundancy until the problem of *Coral 1* is fixed.

The average reallocation time for each step for scenario 1 and 2 is measured and is shown in Figure 4. The time for ”node unavailable” and ”confirmation” is the time taken to detect that the node is unavailable and to confirm that. As shown in the chart, the time for two scenarios is nearly the same. The only difference is that, it does not cost extra time to pull, create and start the container image for scenario two, because it is already deployed there.

With our setup and configuration, the reallocation of service during runtime is successfully realised and the feasibility of the proposed on-demand redundancy concept is proved. The result shows that, although it is not suitable for real-time and fail-operational functions due to the reallocation time, it could be used to get higher availability for non safety-critical functions if certain device is being attacked and the function it provides becomes unavailable. Since the time for detecting unavailable nodes and the confirmation of that can be tuned by changing

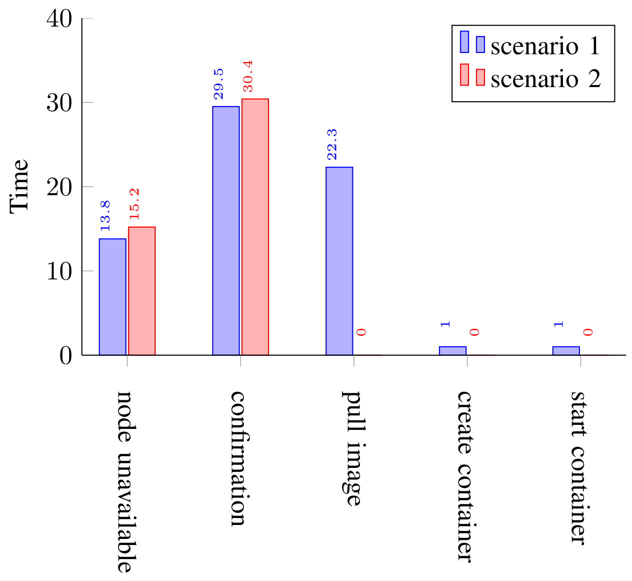

Fig. 4. Evaluation Result

parameters of the container orchestrator k3s, the concept could also be extended in the future for soft real-time functions.

V. Summary and Conclusion

Higher connectivity and more flexible architecture made many novel autonomous vehicle functions possible. The security issues as a side effect must be considered during both the design and runtime phase, as certain services become unavailable because of cyber-attack and affect the driving experience.

In this work, we propose an on-demand redundancy concept using a flexible microservice architecture to enhance the availability of non-safety components in future autonomous driving vehicles. A dynamic microservice architecture is defined, which enables the proposed redundancy concept. Taking advantage of the dynamic aspects of this flexible architecture, the process of the on-demand redundancy concept is described. The proposed concept was implemented and evaluated on different connected devices by leveraging current container technology with orchestration possibilities. Services could be deployed as a container and be scheduled with a container orchestrator. This enables the migration of services at runtime to other resources during cyber-attacks or hardware/software failure. With our implemented setup, the feasibility of the proposed concept is proved. A performance evaluation was also conducted. The result showed that the on-demand redundancy could be set up dynamically at runtime to get higher function availability and could be taken as a security measure against cyber-attacks.

In the future, we plan to adjust the concept and also prove the feasibility for other domains. The latency of the service reallocation process could also be optimized so that the concept could also be extended for functions with soft real-time requirement. We are also considering to evaluate the concept with future automotive hardware setup.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research (BMBF) with funding number 16KIS0885K.

References

[1] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang and W. Shi, ”Edge Computing for Autonomous Driving: Opportunities and Challenges,” in Proceedings of the IEEE, vol. 107, no. 8, pp. 1697-1716, Aug. 2019, doi: 10.1109/JPROC.2019.2915983.

[2] Johansson, R., Andersson, R., Dernevik, M. (2018, January). Enabling Tomorrow’s Road Vehicles by Service-Oriented Platform Patterns.

[3] S. Kugele, D. Hettler and J. Peter, ”Data-Centric Communication and Containerization for Future Automotive Software Architectures,” 2018 IEEE International Conference on Software Architecture (ICSA), Seattle, WA, 2018, pp. 65-6509, doi: 10.1109/ICSA.2018.00016.

[4] S. Behere and M. Torngren, ”A functional architecture for autonomous driving,” 2015 First International Workshop on Automotive Software Architecture (WASA), Montreal, QC, 2015, pp. 3-10, doi: 10.1145/2752489.2752491.

[5] Kugele, S., Hettler, D., Shafaei, S. (2018, November). Elastic service provision for intelligent vehicle functions. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC) (pp. 3183-3190). IEEE.

[6] Kampmann, A., Alrifaee, B., Kohout, M., Wüstenberg, A., Woopen, T., Nolte, M., Kowalewski, S. (2019, October). A dynamic service-oriented software architecture for highly automated vehicles. In 2019 IEEE Intelligent Transportation Systems Conference (ITSC) (pp. 2101-2108). IEEE.

[7] Aoyama, M., Hamano, S. (2019). A Design Methodology of Automotive Service-Oriented Architecture and its Evaluation. Transactions of Society of Automotive Engineers of Japan, 76(1).

[8] Philipp, O., Stefan, K., Eric, S. (2019, September). Model-based resource analysis and synthesis of service-oriented automotive software architectures. In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS) (pp. 128-138). IEEE.

[9] Lotz, Jannik, et al. ”Microservice Architectures for Advanced Driver Assistance Systems: A Case-Study.” 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, 2019.

[10] Wang, Y., Bao, Q. (2020, January). Adapting a Container Infrastructure for Autonomous Vehicle Development. In 2020 10th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0182-0187). IEEE.

[11] Berger, C., Nguyen, B., Benderius, O. (2017, April). Containerized development and microservices for self-driving vehicles: experiences best practices. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW) (pp. 7-12). IEEE.

[12] F. Oszwald, P. Obergfell, M. Traub and J. Becker, ”Reliable Fail-Operational Automotive E/E-Architectures by Dynamic Redundancy and Reconfiguration,” 2019 32nd IEEE International System-on-Chip Conference (SOCC), Singapore, 2019, pp. 203-208, doi: 10.1109/SOCC46988.2019.1570547977.

[13] F. Oszwald, P. Obergfell, B. Liu, V. Pazmino Betancourt. et al., ”Model-Based Design of Service-Oriented Architectures for Reliable Dynamic Reconfiguration,” SAE Technical Paper 2020-01-1364, 2020, https://doi.org/10.4271/2020-01-1364.

[14] J. Caplan, Z. Al-bayati, H. Zeng and B. H. Meyer, ”Mapping and Scheduling Mixed-Criticality Systems with On-Demand Redundancy,” in IEEE Transactions on Computers, vol. 67, no. 4, pp. 582-588, 1 April 2018, doi: 10.1109/TC.2017.2762293.