Opening up New Fail-safe Layers in Distributed Vehicle Computer Systems

Johannes Büttner, Markus Kucera and Thomas Waas
Ostbayerische Technische Hochschule Regensburg,
Faculty of Computer Science and Mathematics,
93053 Regensburg, Germany

Keywords: Automotive, System Architecture, Safety Against Failure.

Abstract: The automotive industry currently faces several challenges, including a growing complexity in system architecture. At the same time, the task load as well as the needs for performance increase. To address this problem, the A3F[a] research project evaluates scalable distributed concepts for future vehicle system architectures. These can be seen as comparable to cluster-computing systems, which are applied in high-performance or high-availability use-cases. Methods used in such scenarios will also be important features in future vehicle architectures such as horizontal application scalability, application load balancing and reallocation, as well as functionality upgrades triggered by the user.
This paper focuses on concepts and methods for the reliability of applications and hardware in future in-vehicle distributed system architectures. It is argued that future automotive computing systems will evolve towards enterprise IT systems similar to today's data centers. Furthermore, it is stated these vehicle systems can benefit greatly from IT systems.
In particular, the safety against failure of functions and hardware in such systems is discussed. For this purpose, various of such mechanisms used in information technology are investigated. A layer-based classification is proposed, representing the different fail-safe levels.

1 INTRODUCTION

Digitilisation has brought about enormous change in many industrial sectors over the last two decades. Through a higher online presence of individuals and the associated expansion of IT infrastructures, such as cloud computing services providers have been using highly scalable, distributed systems productively for years in order to ensure high availability and high performance of their services – even under strongly fluctuating user numbers. Important features of such systems are flexibility and the possibility of continuous further development of both the applications offered and the infrastructures themselves (upgradeability).

On the other hand, the size and complexity of control software is increasing in the automotive industry, as (Reinhardt et al., 2016) notes. This is accompanied by the need for increasing calculation speeds and a higher communication bandwidth between the software components. However, the heterogeneous network infrastructures in current vehicles is reaching their limits (Weckemann, 2014). Assuming that future connected cars as well as automated driving vehicles will require even more communication bandwidth, the current network infrastructure is not a profitable option for such vehicles.

In addition, changing user expectations demand flexible architectural patterns and upgradeability without the need to visit a garage. This establishes new business models ("pay-per-use"). Instead of purchasing special equipment-functions which require their own ECU, customers should be able to subscribe to or unsubscribe from that function in the future on demand and without the need for costly hardware interventions. Similar business models have been used successfully in enterprise IT and cloud computing for years. This also enables manufacturers to integrate security-relevant, error-repairing or simply function-enhancing software updates with little effort. However, the current, statically developed and configured ECU topology does not offer any practicable possibilities for this.

[a] "Ausfallsichere Architekturen für Autonome Fahrzeuge" – fail-safe architectures for autonomous driving vehicles

Büttner, J., Kucera, M. and Waas, T.
Opening up New Fail-safe Layers in Distributed Vehicle Computer Systems.
DOI: 10.5220/0006903600980102
In *Proceedings of the 8th International Joint Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2018)*, pages 98-102
ISBN: 978-989-758-322-3

The desired flexibility requires a fundamental revision and redesign of the actual in-vehicle system architectures. This is accompanied by the conversion of structures and development processes of the corresponding manufacturers (Conway, 1968). This trend can often be found under the heading "service-oriented architecture". A new type of system architecture is expected to meet future requirements in terms of space, cost, performance and energy consumption of the computing units in the vehicle, which arise on the one hand as a result of applications such as automated driving, and on the other hand as a result of changed user expectations.

Flexible, service-oriented architectural patterns have proven themselves in information technology for years. With the increasing spread of key technologies such as automotive Ethernet and the use of service-oriented architectures in vehicles, the synergy potential between these sectors is increasing (Doherty et al., 2004). Therefore it naturally comes to mind to investigate technologies and concepts from the IT sector and to evaluate their usability in the vehicle.

2 APPROACH OF THIS RESEARCH

A similar development has taken place in the field of enterprise IT (cluster computing). Technologies such as Ethernet, virtualization and flexible software architectures have proven themselves here for years. Relevant infrastructures, for example from cloud computing providers have commonly been used in high-performance or high-availability applications. In the research project A3F we investigate which of these concepts and methods can be applied to modern vehicle system architectures. Among other things, the aim is to assess the synergy potential of the two sectors information technology and automotive industry, which to date have very different orientations. However, this synergy is expected to grow strongly in the course of the developments mentioned above.

The aforementioned investigations are carried out using a test cluster. In the following, the hardware and software of this cluster and the reasons for their selection will be discussed briefly.

2.1 Hardware

Many applications of modern in-vehicle functions primarily require high processing speeds, but are not dependent on specific surrounding hardware and thus can be executed on generic processors. Examples of this are multimedia applications, algorithms for image processing or calculations of optimal vehicle speeds and routes.

In the A3F project, a computer cluster consisting of sev- eral nodes is proposed, on which performance-demanding and computation-intensive applications are executed, e.g. the calculation of optimal vehicle speeds and routes, but no real-time control functions.

We believe that such functions as well as special control devices for actuators, sensors and possible gateways for bus systems will continue to exist and remain connected to the computer cluster. The functions running on such a cluster should be executable on almost any node.

There are several reasons why this is desirable:

- *scalablility*: It becomes easy to implement vehicle platform equipment with a variable number of computing units, which can be higher in premium vehicles than in cheaper models, for example. At the same time, it is possible to make simple changes to this equipment if required, for example if a customer wishes to purchase additional features and the existing hardware is no longer sufficient.
- *safety against failures*: Certain functions must be highly available for security reasons. These can be executed redundantly on various nodes. Furthermore, in case of hardware errors, software functions can be moved to other nodes.
- *performance*: It is expected that a computer cluster will provide enough computing power to solve the tasks faced in autonomous vehicles.
- *independence of manufacturers*: By using generic hardware units, these are interchangeable and manufacturers no longer have to keep their control units in stock for decades.
- *upgradeability*: By enabling dynamic allocation of functions to nodes, both hardware and software can be easily exchanged, added or upgraded.

The proposed system consists of several Intel NUC-Kits. These are often used in the relevant fields as examples of homogeneous, powerful but generic hardware units with high resources (e.g. CPU, RAM). They are connected via an Ethernet network.

2.2 Software

The architecture examined in this project should make it possible to flexibly retrofit software updates into the computer cluster. It should therefore be possible to run individual applications on any node in the cluster, largely independently of specific hardware. In addition, several software functions should be able to be executed on individual hardware units. In addition, it is evaluated whether and how functions can be added

at runtime, moved within the cluster or updated (for security updates, for example).

As frequently seen in cluster computing architectures, the overall system is designed such that applications are not tightly coupled to a specific computing unit. Instead, we use a container engine to encapsulate applications with their runtime environment and dependencies. In this way, the infrastructure gains the necessary flexibility so that live migrations of applications between different computing units can be carried out. Moreover, this provides an easy way to monitor applications per process as each container typically includes only one process. Finally, containers offer a simple way to limit the resource consumption of an application.

This step implies a fundamental redesign of the affected software and ECU architecture of todays vehicles, as some basic assumptions of traditional E/E architectures become invalid. At the same time, the shift towards distributed systems took place years ago in a similar form in enterprise IT and cloud computing infrastructures. The extent to which applied technologies and concepts can be applied in modern vehicle architectures will therefore be examined.

However, this requires a precise knowledge and examination of the technical details and problems of both sectors. Since safety is a high priority in the automotive sector, an overview of the various possibilities for implementing safety against failures in IT server systems is presented in this paper.

3 FAILOVER STRATEGIES

Looking at the various fail-safe mechanisms of server systems used in information technology, one will first realize that there are a multitude of possibilities for realizing this. Specifically, as depicted in 1, such systems can be divided into several logical layers, and there are different approaches and procedures, depending on the level at which failover is implemented. Therefore, the different layers of such a system will be discussed in the following.

3.1 Logical Layers of Distributed Systems

The layers mentioned above are in principle layers of abstraction designed to ensure interoperability and interchangeability. Fixed interfaces are defined for the levels adjacent above and below, so that the concrete implementation of a level only has to adhere to these interfaces. First of all, it should be mentioned that any

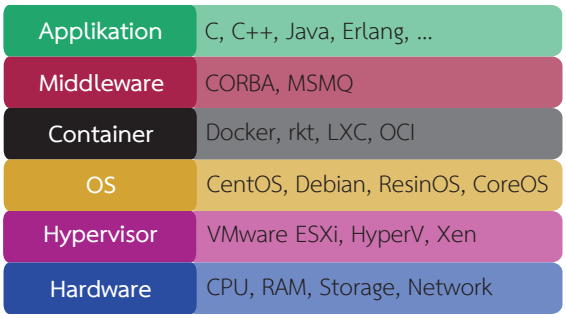

Figure 1: Logical Layers of Distributed Systems, with examples.

number of layers can be defined, which - depending on the complexity of the respective application case - represent a reasonable abstraction or an unnecessary complication. An example of a proven layer model is the well-known Open Systems Interconnection model (OSI model). It defines seven layers with which communication between computing units in computer networks can take place. This model is standardized and is used to cope with the complexity of communication systems.

Such a model can also be defined for server systems, although there is no standardized form here. However, specified interfaces between the layers do exist. In the course of this work the different levels are needed for a classification.

Hardware. The lowest level represents the physical hardware of a computer system, such as CPU, memory or non-volatile memory. Connections to peripheral devices and the network infrastructure also belong to this level.

Hypervisor. On server systems, virtual machines have been used almost exclusively for several years. These allow the often generously dimensioned resources of a server to be divided into smaller segments, each equipped with its own operating system and fulfilling its own task. In this way, several small servers, each of which previously required its own hardware, are merged on more powerful servers. A hypervisor is used here as the administration instance. This is shown schematically in figure 2.

Operating System. Servers today mostly use Linux-based operating systems. Applications or programs are then programmed against a special standardized interface (e.g. POSIX). One of the most important tasks of an operating system in this context is to run programs largely hardware-independent. An operating system must therefore have an exact knowledge of the hardware on which it was installed.

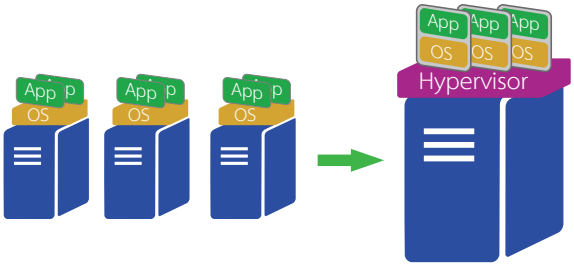

Figure 2: Consolidation of multiple small servers on powerful servers with hypervisor.

Container. With the arrival of web-distributed systems and applications that can be flexibly scaled to between a thousand and ten thousand instances depending on the load, the virtual machine is once again of lesser importance because it is too heavy to guarantee this type of flexibility. Examples include e-commerce systems such as eBay or amazon, or media streaming services such as Netflix or Spotify. To improve customer satisfaction, they strive for constant availability of their services - even under strongly fluctuating user numbers.

Lighter virtualization technologies, called containers, have therefore become established for such applications. As opposed to virtual machines, these do not require their own operating system. Such containers can be started up at high speed on computers in the event of an increase in user requests. Corresponding investigations(Xavier et al., 2013) have shown that container technology is superior to the virtual machine in its performance. However, some questions remain unanswered with regard to security aspects(Mohallel et al., 2016) that do not arise with a virtual machine.

Middleware. Middleware refers to one or more abstraction layers that ensure interoperability and transparency. The exact definition is seen differently in the literature. It is typically intended to hide the underlying infrastructure from application developers, thus relieving them from paying attention to certain technical details, so that they can concentrate on the functionalities. It can also add additional technical, non-functional features.

There are different types of middleware. On the one hand, there is the Remoting Middleware, which enables method calls via the network. Examples are CORBA, Java RMI and .NET Remoting. On the other hand, there is the messaging middleware, which deals with communication with other components via messages. Examples are MSMQ or MQSeries; there are also standards such as MQTT.

Application. The application level represents the highest level in a distributed server system and contains the actual logic and functionality. Such applications are written in a programming language like C, C++ or Java. They are closely linked to the operating system on which they are to be executed.

After this classification, we examine at which level existing fail-safe mechanisms are implemented and which advantages and disadvantages result from this.

3.2 Fail-safe Layers

There are various possibilities to realize safety against failure. Characteristic for the strategies presented here is that the task of one application is taken over by a second (backup) application in the event of an error, so that the overall system never fails. This strategy is often referred to as „fail-operational“. It differs from conventional strategies, which often only provide for the safety-critical function to be switched off in the event of a fault (fail-safe). It is assumed here that the correct function of an automotive ECU can be guaranteed by maintaining it - in the event of an error or component failure - by a second, redundantly running ECU. In addition, applications should be able to migrate to a second ECU at runtime in order to be able to reconfigure the entire system in the event of an error. This corresponds to the usual procedure in the server environment and is shown in 3.

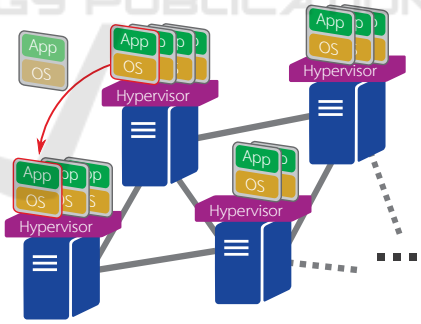

Figure 3: Moving or migrating applications at runtime.

Hypervisor-layer. The hypervisor provider VMware offers “vSphere FT” (VMWare, 2013) (Fault Tolerant), which provides a solution to mirror calculations on one machine redundantly on a second virtual machine in real time. If the active machine fails, the backup machine takes over.

It should be noted that the failure of the first machine must be detected very quickly. Since the solution is designed to meet the requirements of Enterprise IT, in which a few seconds of downtime can cost thousands of dollars but not human lives, it must be

carefully examined whether such technologies can be used in an automotive application with safety-critical software. In addition, the constant alignment of the two machines requires a high communication bandwidth, which, although not a problem in a server environment, is again subject to different requirements (EMC) in the vehicle and cannot necessarily be guaranteed. The strategy of machines running in lockstep, however, offers a high degree of reliability under certain conditions and time requirements. The determination of exact times and concrete measurements is no longer part of this work, but will be carried out in a follow-up examination.

Container-layer. There are also solutions at container level, such as the open source orchestration tool „Kubernetes“ developed by Google, which offers the possibility of running applications in containers redundantly. Containers can encapsulate the executables of the application and all dependencies without sacrificing the performance of operating system virtualization.

However, the disadvantage in terms of failover is the lengthy detection of a failure, which is even longer than the VMware solution described above. In general, the tool is more designed for scalability of services and only offers good reliability mechanisms at first glance. These are not applicable to future automotive control units; however, the container concept could well be used here in general, since they offer a high degree of flexibility with simultaneous small performance losses.

Application-layer. Concepts to ensure the reliability of services at application level were also examined in the context of the work. Programming languages (e.g. Erlang) or application frameworks (e.g. Akka) with corresponding features are used. Application developers use these features to distribute the programs on a network of servers.

This offers the highest degree of flexibility and fine granularity. Only those parts of a program that really need it can be designed redundantly. In addition, the most reliable way to detect a failure is from the application logic. However, the most capable developers are required for this. This type of programming is perceived as particularly difficult, but is ultimately the safest option.

4 CONCLUSION

In this paper, fail-safe mechanisms on different levels of a distributed system were presented. The closer these mechanisms are placed near the hardware, the faster and safer a redundant configuration of several processing units can be implemented. However, in this case the flexibility decreases and the performance overhead increases. The more application-specific a redundancy mechanism is, the lighter and finer granular it can be.

However, redundancy alone is no guarantee for safety against failures, since failure detection cannot offer sufficient coverage when not taking all system's layers into account. Therefore mechanisms to implement this safety measures within all layers of the framework have to be provided.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the financial support by the Bavarian Ministry of Economic Affairs, Energy and Technology, funding progamme “Information and Communication Technology Bavaria” as well as the support by project management organization VDI/VDE Innovation + Technik GmbH.

REFERENCES

Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4):28–31.

Doherty, P., Haslum, P., Heintz, F., Merz, T., Nyblom, P., Persson, T., and Wingman, B. (2004). A distributed architecture for autonomous unmanned aerial vehicle experimentation. In *Distributed Autonomous Robotic Systems 6*, pages 233–242. Springer.

Mohallel, A. A., Bass, J. M., and Dehghantaha, A. (2016). Experimenting with docker: Linux container and base OS attack surfaces. In *2016 International Conference on Information Society (i-Society)*, pages 17–21.

Reinhardt, D., Kühnhauser, W., Baumgarten, U., and Kucera, M. (2016). *Virtualisierung eingebetteter Echtzeitsysteme im Mehrkernbetrieb zur Partitionierung sicherheitsrelevanter Fahrzeugsoftware*. Universitätsverlag Ilmenau, Ilmenau. OCLC: 951392623.

VMWare (2013). Wie die Fault Tolerance funktionert.

Weckemann, K. (2014). *Domänenübergreifende Anwendungskommunikation im IP-basierten Fahrzeugbordnetz*. PhD thesis, lmu.

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and De Rose, C. A. F. (2013). Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. pages 233–240. IEEE.