Resource Allocation in Software-Defined Vehicles: ILP Model Formulation and Solver Evaluation

Fengjunjie Pan, Jianjie Lin, Markus Rickert, and Alois Knoll

***Abstract*— Modern vehicles are changing rapidly. Years ago, cars were still differentiated from each other by mechanical features such as horsepower. Nowadays, there are nearly 100 electronic control units (ECUs) and millions of lines of codes embedded in a vehicle. In the automotive industry, software is starting to have more value than hardware. Thus, the concept of software-defined vehicles (SDVs) is rising. Traditional electrical and electronic (E/E) architectures show their limitations with predefined implementations, layout and distribution, as well as insufficient computational power for future extensions. Instead, powerful automotive central computing platforms with flexible software solutions are being developed. With the development of hypervisor technology, applications with different safety criticality can be deployed on different virtual machines (VMs) of the same physical hardware. The resource allocation problem, which refers to mapping software to hardware, is becoming more complicated with the increasing number of automotive applications. Existing approaches for resource allocation problems mainly focus on distributed E/E architectures. In this paper, we aim to address the gap of solving resource allocation problems in the central computing platform of SDVs, where the dynamical VM creation and constrained application deployment shall be considered simultaneously. We propose an Integer Linear Programming (ILP) based approach by formulating the problem as an optimization problem with the minimum number of VMs as the goal and utilizing well-known solvers to find the solution automatically. Moreover, we evaluate the performance of state-of-the-art solvers for the proposed approach.**

I. Introduction

Driven by the complexity of autonomous driving and its rapid development cycle, the focus of the automotive industry is shifting more and more toward software development. Where automotive architectures were previously determined by a predefined number of electronic control units (ECUs) with fixed feature sets, changes in autonomous driving algorithms and constant updates are paving the way for software-driven architectures on powerful and flexible hardware platforms. Instead of complex upgrades to a vehicle that could only be performed at a local dealership, software-defined vehicles (SDV) can be seamlessly updated and upgraded over the air [1], [2].

Centralized electrical and electronic (E/E) architectures with high-performance computers are therefore becoming the trend [3]. They provide flexibility, scalability, compatibility, and upgradeability for SDVs. Vehicles are complex systems consisting of applications with mixed-criticality. The ISO 26262 standard defines requirements for freedom from interference among applications with different safety levels [4]. Concepts such as hypervisors are introduced in order to fulfill these requirements in multi-core systems [5]. Fig. 1 shows the structure of a high performance central computational platform inside a centralized E/E architecture. The central computing platform is divided into separate environments by using virtualization technology, which enables applications with mixed-criticality levels to run on the same physical hardware. In general, the characteristics of SDVs can be described as follows: first, software becomes centralized by employing a high-performance computer; second, software and hardware are decoupled, enabling flexible deployment; third, hardware is divided by the hypervisor, ensuring the safety requirements of vehicles [6], [7], [8], [9], [10].

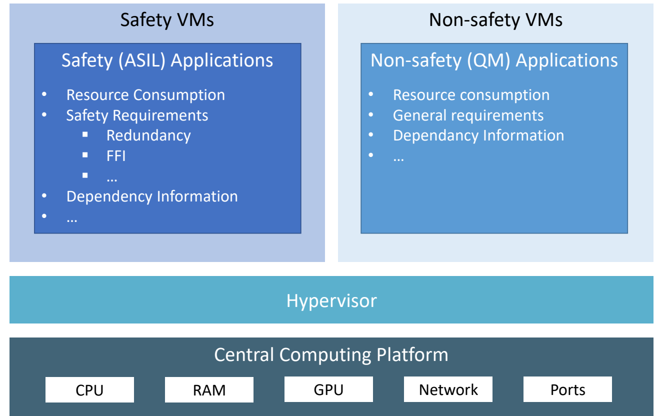


Fig. 1: Structure of the central computing platform. The central platform is separated into different VMs by the hypervisor, providing execution environments for mixed-critical applications.

Given these complex requirements and more flexible hardware platforms, the question of optimal resource allocation has to be addressed. The goal is to find an optimal allocation of a fixed amount of resources to activities while minimizing the allocation cost [11]. In current automotive engineering, resource allocation, specifically mapping applications onto hardware, is a manual process that requires a large amount of experience. However, given shorter update cycles and an increasing number of applications, it will become more critical and soon unmanageable. In addition, the inter-dependence of applications and their requirements introduces even more challenges. Many approaches exist in the literature [12], [13],

F. Pan, J. Lin, M. Rickert, and A. Knoll are with Robotics, Artificial Intelligence and Real-Time Systems, Department of Informatics, Technical University of Munich, Munich, Germany. {panf, jianjie.lin, rickert, knoll}@in.tum.de

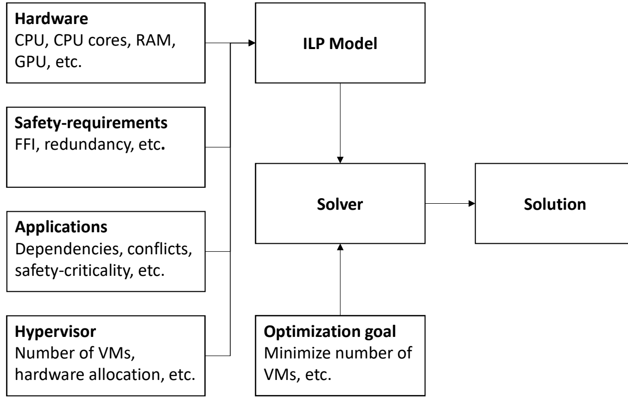


Fig. 2: Proposed method of solving resource allocation problems. Related information and requirements are formulated as an ILP model, which will be sent to the solver. The solver will then search for the (optimum) mapping decisions accordingly. The solution details are presented in Fig. 3.

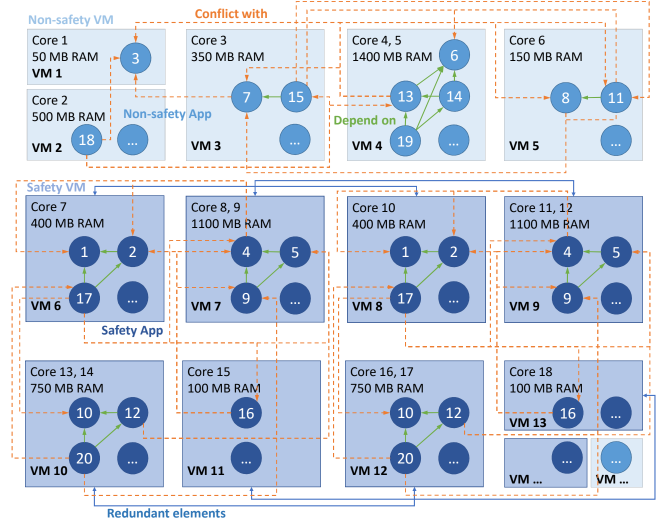


Fig. 3: Partial illustration of the optimal solution from one of the generated resource allocation problems. Cores and memory are allocated to each safety or non-safety VM (dark/light blue squares). Safety and non-safety applications (dark/light blue circles) are allocated accordingly to VMs. The green, orange and blue lines with arrows represent respectively the dependency, conflict, and redundancy relationships among applications.

[14] for finding automotive resource allocation solutions for distributed E/E architecture, where the ECUs have a fixed number and unchangeable properties. However, in SDVs with a flexible central computing platform, the virtual machines (VMs) shall be allocated on demand dynamically. In this paper, we address the problem of resource allocation for SDVs with the objective goal of minimizing the number of VMs under various constraints. Our proposal of automated solving is illustrated in Fig. 2. We take inspiration from the bin-packing problem, in which items of different sizes must be packed into a minimum number of containers [15] and utilize Integer Linear Programming (ILP) formulations. Our work in this paper includes: (i) collecting constraints for resource allocation problems in the SDVs, (ii) proposing an ILP model to formalize the problem, (iii) comparing the performance of solvers in solving the proposed ILP model.

This paper is structured as follows: Section II introduces related works about resource allocation in the automotive and similar domains. Section III and IV discuss related constraints and introduce the proposed ILP formulation. The experimental setup and benchmarking results are presented in Section V, VI. Finally, a conclusion is given in Section VII.

II. Related Work

Our research draws on different literature directions. Firstly, we will discuss the related method in automotive resource allocation problems, in which applications are assigned to different hosts according to various constraints. Secondly, we will introduce related literature on bin packing in cloud computing, which helps dynamically allocate resources to VMs.

Mehiaoui-Hamitou et al. [16] presented a mixed-integer linear programming (MILP) based approach to solving the automotive deployment and scheduling problem regarding energy efficiency. They considered constraints including timing and utilization values to map tasks on ECUs and considered power consumption for computation and communication as the cost function to enable shutdown of an ECU if no tasks are needed in a specific driving situation. Zhang et al. [12] provided MILP and Genetic algorithm (GA) based methods to map software components on ECUs. This paper exploits legacy architectures and the AUTomotive Open System ARchitecture (AUTOSAR) standard to reduce the design space and optimize the computation time. They defined their mapping constraints as: dedicated/exclusive mapping (certain software components must/cannot be assigned to specific ECUs), component separation/clustering (certain software components must/cannot run on the same ECU). Voss et al. [13] discussed a joint problem of scheduling and deployment for mixed-criticality multicore architectures considering core numbers and safety levels of tasks. They formulated the problem as a satisfiability problem utilizing SMT solvers. Maticu et al. [14] discussed the mapping of applications with mixed-criticality to AUTOSAR distributed multicore architectures with consideration of AUTOSAR constraints (e.g., elements of the same software component must be assigned on the same ECU), schedulability, and safe criticality (e.g., runnables with different safety level cannot be mapped together). A simulated annealing-based meta-heuristic optimization approach was proposed. Pohlmann et al. [17] proposed a model-driven approach for specifying allocation problems and finding feasible allocations. They developed a domain-specific language (DSL) and defined allocation constraint types in distributed E/E architecture as

TABLE I: Summary of Notations

Symbol	Meaning
$\mathcal{A}$	Set of applications
$\mathcal{A}_{\text{safe}}$	Set of safety-critical applications. $\mathcal{A}_{\text{safe}} \subset \mathcal{A}$
$\mathcal{A}_{\text{nsafe}}$	Set of non-safety-critical applications. $\mathcal{A}_{\text{nsafe}} \subset \mathcal{A}$
$\mathcal{A}^a_{\text{dep}}$	Set of dependent applications of $a \in \mathcal{A}$, $\mathcal{A}^a_{\text{dep}} \subset \mathcal{A}$
$\mathcal{A}^a_{\text{conf}}$	Set of conflict applications of $a \in \mathcal{A}$, $\mathcal{A}^a_{\text{conf}} \subset \mathcal{A}$
$\mathcal{V}$	Set of available VMs
R	Integer value expressing the total amount of available resources as quantitative numbers, e.g., number of CPU cores, number of GPUs, size of memory.
a, a'	Application $a, a' \in \mathcal{A}$
v	VM $v \in \mathcal{V}$
d_v	Binary variable depicting if VM v is allocated with any resource and ready for use.
r_a	Integer value representing application a's consumption of quantitative resources, e.g., number of CPU cores, number of GPUs, size of memory.
r_v	Integer variable representing amount of allocated resources in VM v, e.g., number of CPU cores, number of GPUs, size of memory.
s_a	Binary variable representing safety criticality of application a. If $a \in \mathcal{A}_{\text{safe}}$, then $s_a = 1$. If $a \in \mathcal{A}_{\text{nsafe}}$, then $s_a = 0$.
s_v	Binary variable representing safety criticality of VM v. $s_v = 1$ means v is safety-critical. $s_v = 0$ means v is non-safety-critical.
$x_{a \to v}$	Binary variables depicting if applications a is mapped on VM v

follows: col/separate-location (two components have to be allocated to the same or different ECUs), required location (a component has to be allocated to a specific ECU), required resource (e.g., memory, task-scheduling). Then, they transformed the DSL to an ILP representation to find feasible solutions.

The literature mentioned above provides us with diverse aspects of solving the automotive resource allocation problem considering different requirements with/without optimization goals. However, the work discussed so far mainly focused on distributed E/E architecture, where ECUs have fixed numbers and sizes. On the other hand, SDVs have more dynamic environments, where VMs need to be allocated according to the application requirements. Therefore, we further investigated the bin packing problem in cloud computing. The bin packing problem can be described as assigning multiple items to the minimum number of bins. Fatima et al. [18] formulated the VM placement problem as a variable-sized bin packing problem (VSBPP), where bins have different capacities and costs. Kaaouache et al. [19] adapted their hybrid GA method of the bin packing problem to solve VM placement in the cloud.

In this paper, we consider the resource allocation problem in the context of SDVs, where central computing platforms and VMs are employed. We set the optimization goal similar to the bin packing problem to minimize the VM numbers.

III. Problem Description

The resource allocation in this paper contains two steps. Firstly, VMs with sufficient resources must be installed in the central computing platform. Then, applications should be mapped onto VMs under the consideration of constraints. We aim to find the minimal number of required VMs. We primarily consider the constraints in the following four directions for automotive application deployment: resource availability, freedom from interference, redundancy, and dependency.

Resource availability restricts the possibilities of arbitrary resource assignment in the automotive domain. In the context of a software-defined vehicle, the hardware resources in a vehicle are mainly core, memory, and storage, which are essential for executing an application. Besides, a predefined amount of resources are required to execute an application. As illustrated in Fig. 1, more than one application can be deployed inside a virtual machine. Therefore, resource allocation to each virtual machine is the first task to be addressed.

Freedom from interference (FFI) is a constraint defined in ISO 26262 [4]. It is defined as there being no cascading failures between two or more elements that could lead to a violation of safety requirements. An element can be a system or part of the system, including hardware and software. Cascading failures mean that the failure of one element of an item leads to the failure of one or more elements of the same item. In the context of SDVs, we primarily consider the failure of software. According to the ISO standard, the vehicular software can be classified into different safety levels: QM for non-safety-critical applications, such as infotainment systems, ASIL A, B, C and D for safety-critical applications, such as object detection. Based on these observations, we define FFI constraints that allow only one virtual machine to install applications with the same level of safety and require the same level of safety to be present in that virtual machine. For the ease of describing and demonstrating the proposed approach, we simplify the safety properties as safety-critical and non-safety-critical. However, the extension to consider complete safety levels is discussed in the following sections.

Redundancy is essential when designing safety-critical systems. The goal is to equip a system with multiple components or subsystems that perform the same function so that if one of them fails unexpectedly, its redundant devices can take over immediately and ensure that the entire system can still safely perform its assigned tasks. For example, an autonomous driving system is a safety-critical system. The constraint defined in redundancy is to equip a replica for each safety-critical application, which runs in a different virtual machine.

Dependency constraints describe the relationship between interdependencies and conflicts of applications. If applications have dependencies on each other, they should be assigned to the same virtual machine. Conversely, if two applications have conflicts, they should be placed in different virtual machines.

IV. ILP Model Formulation

We formulate the hardware-software resource allocation problem as an ILP model. In this work, we focus our efforts on the four essential constraints mentioned above, which

can cover most cases. At the same time, new constraint types as well as optimization goals (e.g., minimize the number of allocated cores) can easily be integrated into this optimization model if they can fulfill the ILP criteria. The related notations are defined in Table I. We formulate the resource allocation in SDVs as follows:

$$\text{minimize: } \sum_{v \in \mathcal{V}} d_v \tag{1}$$

$$\text{subject to: } \sum_{v \in \mathcal{V}} r_v \leq R \tag{2}$$

$$0 \leq r_v \leq R d_v, \forall v \in \mathcal{V} \tag{3}$$

$$\sum_{a \in \mathcal{A}} r_a x_{a \to v} \leq r_v, \forall v \in \mathcal{V} \tag{4}$$

$$\sum_{v \in \mathcal{V}} x_{a \to v} = 1 + s_a, \forall a \in \mathcal{A} \tag{5}$$

$$x_{a \to v} + s_v \leq 1, \forall a \in \mathcal{A}_{\text{nsafe}}, v \in \mathcal{V} \tag{6}$$

$$x_{a \to v} - s_v \leq 0, \forall a \in \mathcal{A}_{\text{safe}}, v \in \mathcal{V} \tag{7}$$

$$x_{a \to v} = x_{a' \to v}, \forall a \in \mathcal{A}, a' \in \mathcal{A}^a_{\text{dep}}, v \in \mathcal{V} \tag{8}$$

$$x_{a \to v} + x_{a' \to v} \leq 1, \forall a \in \mathcal{A}, a' \in \mathcal{A}^a_{\text{conf}}, v \in \mathcal{V} \tag{9}$$

We consider $\mathcal{A}$, $\mathcal{A}_{\text{safe}}$, $\mathcal{A}_{\text{nsafe}}$, $\mathcal{A}^a_{\text{dep}}$, $\mathcal{A}^a_{\text{conf}}$, r_a, s_a as input information from applications, R as the size of known hardware, and $\mathcal{V}$ as the set of maximum available VMs. The number of elements in $\mathcal{V}$ is limited by the hardware resources and capabilities of hypervisors, e.g., if one single core cannot be split into different VMs, this implies that the number of VMs is smaller than the number of cores. The VM v is allocated with resources and can host applications when d_v equals 1. Decision variables of solvers are d_v, r_v, s_v, $x_{a \to v}$.

In our ILP model, (1) guarantees that a minimum number of VMs are used. (2) restricts that the total resources of all VMs should not exceed the capability of physical hardware resources. (3) implies that only when the specific VM is used ($d_v > 0$), resources can be allocated to it ($r_v > 0$). (4) describes that each VM should provide enough resources for all applications assigned to it. Following the logic of (2), (3), (4), heterogeneous resources have to be expressed separately. (5) relates to the redundancy constraint. Non-safety-critical applications should be assigned only once, while safety-critical applications should have a replica in a different VM. (6) and (7) guarantee FFI. VMs should have the same safety properties as allocated applications. It also implies that applications with different safety levels cannot be placed in the same VM. Both of the expressions utilize linear classifiers to guarantee a valid design space. Additional linear decision boundaries can be added to extend the formulation with complete ASIL levels. (8) and (9) introduce dependency constraints. If application a is dependent on a', a should be assigned to the VM, where a' runs. If application a has conflicts with a', they need to be put into different VMs.

V. Experimental Setup

The formulated ILP problem is implemented within a Python environment and the feasible solution and its optimal solution are solved using five state-of-the-art solvers. We compared the performance of different solvers with respect to the calculation time for our problem formulation. To remove ambiguity, the solving process is represented as finding a feasible solution that satisfies the constraints but does not consider any objective function. The optimization process is above the solving process and finds a globally optimal solution in terms of a predefined objective function by obeying constraints. Based on the yearly ILP benchmark competition [20], we selected the following solvers for our evaluation. We summarize the characteristics of each solver based on the following criteria: whether the solver is open source, which programming problem is supported, and which programming languages are supported.

IBM ILOG CPLEX Optimization Studio (CPLEX) 20.1.0 is a commercial optimizer developed by IBM for solving linear, mixed-integer, and quadratic programming [21]. It supports various algorithms for solving linear programming problems, such as primal/dual variants of simplex or barrier interior point methods. CPLEX provides several programming interfaces: C/C++, Java, and Python. In the experiments we used the Python API DOcplex for object-oriented modeling and mathematical programming.

Cardinal Optimizer (COPT) 4.0.2 is a commercial mathematical optimization solver that aims at solving large-scale optimization problems. It provides a high-performance solver for LP, MIP, SOCP, convex QP, and convex QCP [22]. It offers the following programming interfaces: C/C++, Java, and Python. In our experiments, we utilized the Python API to evaluate its performance.

Gurobi 9.5.0 is a commercial mathematical programming solver that is designed from the ground up to exploit modern architectures and multi-core processors, using the most advanced implementations of the latest algorithms such as LP, MILP et.at. [23]. It provides many different interfaces and is easy to be deployed in the workflow.

MOSEK 9.3.14 is a commercial optimization software solving linear, quadratic, semidefinite, and mixed integer problems. We utilized Fusion API to build our model in an expressive manner using mainstream programming languages [24].

SCIP (Solving Constraint Integer Programs) 8.0.0 is a non-commercial solver [25], and works as a general framework based on branching for constraint integer and mixed-integer programming using branch-cut-and-price. It can be used to solve the convex and nonconvex MILP/MINLP problem by utilizing polyhedral outer approximations and a spatial branch and bound technique. This solver uses the LP relaxation and cutting planes to enable a strong dual bound while utilizing the constraint programming to handle arbitrary constraints. We used the extendable Python interface PySCIPOpt [26] in the experiment for evaluating the performance.

We took ADLINK's in-vehicle platform [27] as hardware reference and defined a target platform with 80 cores and 768 GB RAM. Our pre-experiments showed that defining problems with varying application resource usage had a

minor impact on the calculation time of solvers. Thus, to simplify the problem definition, we defined each application to be deployed as requiring 5% of one single-core and 50 MB RAM. The other properties of applications were generated randomly. Generated applications were either defined as safety-critical or non-safety-critical. In addition, each of them was randomly dependent or/and in conflict with other applications. The generated problems had different application numbers varying from 100 to 800 increased by 100. We generated three random application sets for each problem. All generated problems and application sets were independent of each other. In addition, the properties and relationships of applications were guaranteed to be valid and solvable. We conducted both solving and optimization analyses with the same problems generated. Moreover, we performed infeasibility analysis using the same sets of applications by swapping their dependence and conflict properties. The input of solvers contained core numbers, RAM size of the hardware platform, core/memory consumption, safety criticality, and interdependence/conflicts of applications. The solvers shall calculate: (1) how many cores and memories should be assigned to each VM; (2) what is the safety property of VMs; (3) which application should be assigned to which VM; (4) optimize the number of VMs that can host all the applications with pre-defined constraints. The size of decision variables was defined as $[\text{VM count} \cdot (4 + \text{application count})]$. Since there were only 80 cores in the whole hardware platform and the FFI requirement shall be fulfilled, the maximal number of available VMs was limited to 80. The complexity of each problem instance, including the number of decision variables, constraints, and minimal solutions fulfilling the requirements, is presented in Table II. Documented values are average values deriving from three randomly generated application sets of respective problems. The column "VMs" shows the minimal number of VMs, which is the objective value in our optimization. The minimal number of used cores for each problem is also listed to show the usage of the central computing platform. Fig. 3 illustrates the optimal solution from one of the generated problems with 100 applications. Fig. 3 illustrates the optimal solution from one of the generated problems with 100 applications. We visualized the assignment of 20 applications and their relationships in detail and presented the allocation of cores and memory in VMs.

VI. Experimental Results

The experiments were performed on an Intel i7-7600U CPU with 2.80 GHz and two cores (four threads), 16 GB RAM, running Windows 10 Enterprise 21H1. Solvers were executed with their default configurations. A time limitation of 200 seconds was defined for each calculation. We conducted the measurements in three steps. Firstly, we measured the calculation time of solvers for finding feasible solutions. Then, we investigated their performance in finding the optimum solution. Furthermore, we recorded their ability to prove infeasibility. We compared the calculation time for solving, optimization, infeasibility proven activities, and analyzed the effect of application number scaling on the calculation time.

TABLE II: Complexity of generated problems. The numbers of variables are fixed values decided by the number of applications and available VMs. The numbers of constraints, minimal VMs, and cores are given as average values deriving from randomly generated application sets of each problem.

Scenario Definition			Minimal Solutions	
Apps	Variables	Constraints	VMs	Cores
100	8320	142982.00	15.33	18.33
200	16320	549988.67	19.67	27.67
300	24320	1228755.33	20.00	36.33
400	32320	2154322.00	21.33	47.67
500	40320	3381382.00	20.00	48.22
600	48320	4826095.00	22.00	57.33
700	56320	6566009.67	23.67	72.00
800	64320	8578082.00	24.33	74.67

A. *Quantitative comparison on the solving performance*

Table III records the average time to find a feasible solution $\bar{t}$ and its sample standard deviation σ for three randomly generated test sets with a different number of apps. The data from the tables show that the tested solvers are almost always able to find feasible solutions within the specified time. We set the highest time limit to be 200 seconds in our experiments. In terms of the shortest solution time, we can observe that Gurobi has the shortest solution time for all sizes of problems. COPT, CPLEX, and SCIP closely follow it. However, MOSEK takes a longer time to compute the solution. It was unable to complete the task in the 800 applications test scenarios. From the solution robustness point of view, we can learn that all solvers have relatively small standard deviations. They perform stable when solving different random sets with the same number of applications. In terms of the growing trend of searching feasible solution time, we find that the more applications, the longer the solver takes to solve. The standard deviation of the solving time also increases with the number of applications.

B. *Quantitative comparison on the optimization performance*

Table IV presents the optimization performance of each solver. In this table, $\bar{t}$ represents the average optimization time for three random test sets for a different number of application scenarios. σ describes the sample standard deviation of the optimization time. $\bar{n}$ records the average number of feasible solutions found by the solver during the optimization process before finding the optimal solution. As mentioned earlier, the Gurobi solver has once again received a positive performance, outperforming several other solvers, followed by CPLEX. In the experiments, both solvers returned the *optimal* state within the specified time limit and obtained the same optimal objective value. The objective value in our scenario represents the minimum number of virtual machines required to achieve the app resource allocation. We use these optimal objective function values obtained from Guorbi as

TABLE III: Comparison of performance for finding feasible solutions in problems with 80 VMs. The column *Apps* lists the number of applications in each problem. $\bar{t}$ represents the average solving time of three random sets in each problem in [s]. σ shows the sample standard deviation of solving time in [s].

Apps	CPLEX $\bar{t} \pm \sigma$	COPT $\bar{t} \pm \sigma$	Gurobi $\bar{t} \pm \sigma$	MOSEK $\bar{t} \pm \sigma$	SCIP $\bar{t} \pm \sigma$
100	0.48±0.08	0.75±0.09	0.33±0.05	0.79±0.09	0.98±0.06
200	2.05±0.02	1.64±0.08	1.49±0.04	4.35±0.48	3.56±0.49
300	6.33±0.20	4.26±0.13	4.52±0.06	13.54±0.53	9.71±0.22
400	14.95±0.53	9.88±0.69	9.63±0.11	30.66±0.74	20.78±0.96
500	30.68±0.28	18.68±0.37	18.00±0.06	59.61±1.06	37.23±0.13
600	55.34±0.97	32.14±0.48	28.60±0.27	101.21±0.80	64.65±1.60
700	86.72±1.44	54.24±1.68	43.06±0.47	162.40±3.88	103.76±1.51
800	143.48±6.61	82.61±5.01	60.15±1.69	- ± -	162.88±2.59

TABLE IV: Comparison of performance for calculating the optimum solution in problems with 80 VMs. The column *Apps* and *Obj* list the number of applications and the average objective value of random sets in each problem. $\bar{t}$ represents the average optimization time of three random sets for each problem in [s]. σ shows the sample standard deviation of optimization time in [s]. $\overline{n}$ represents the average number of discovered sub-optimal solutions during the optimization.

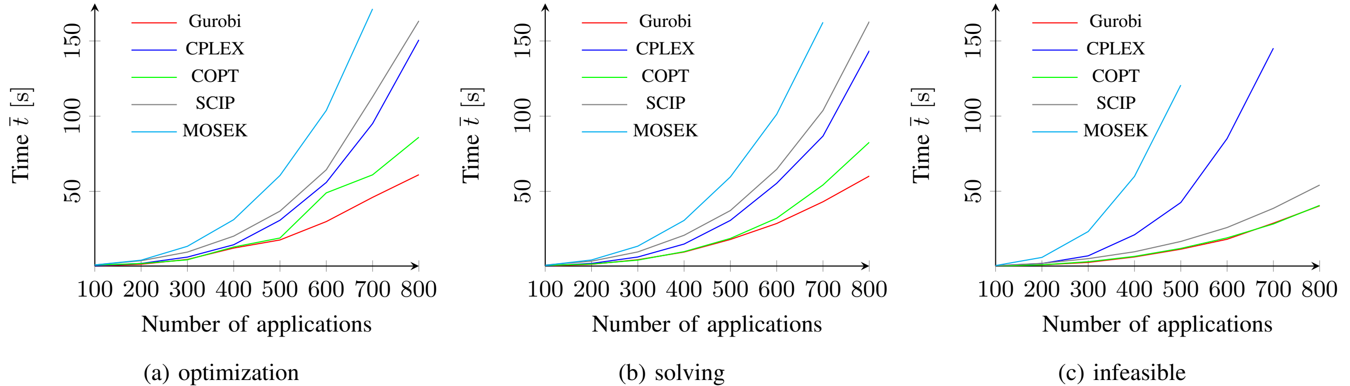
	CPLEX state=optimal[a]		COPT[c] state=feasible[b]		Gurobi state=optimal[a]		MOSEK state=feasible[b]		SCIP state=feasible[b]	
Apps	$\bar{t} \pm \sigma$	$\overline{n}$	$\bar{t}$	$\overline{n}$	$\bar{t} \pm \sigma$	$\overline{n}$	$\bar{t} \pm \sigma$	$\overline{n}$	$\bar{t} \pm \sigma$	$\overline{n}$
100	0.54±0.03	3.00	≤ 1	≤ 4	0.37±0.02	2.00	1.10±0.29	3.00	1.09± 0.18	1.67
200	2.08±0.19	2.00	≤ 2	≤ 3	1.55±0.03	1.33	4.24±0.08	1.00	3.91± 0.08	1.00
300	6.32±0.17	2.00	≤ 5	≤ 7	4.76±0.31	1.00	13.49±0.32	1.00	9.64± 0.19	1.00
400	14.51±0.73	2.67	≤13	≤10	12.32±2.09	1.67	31.19±1.16	1.67	20.24± 0.74	1.00
500	30.81±1.09	2.00	≤19	≤ 8	17.70±3.08	1.33	60.57±1.87	1.00	36.88± 0.31	1.33
600	55.94±1.13	2.33	≤42	≤10	29.92±0.80	1.33	103.80±2.41	1.67	64.36± 0.43	1.00
700	95.06±1.18	2.00	≤60	≤ 6	46.03±3.47	2.33	171.32±9.21	1.67	112.85±15.65	1.33
800	150.73±0.39	2.33	≤84	≤ 5	61.08±1.29	1.33	- ± -	-	163.36± 0.41	1.00

[a] CPLEX and Gurobi were able to determine the optimal solution and return the status *optimal* within pre-set time limit.
[b] COPT, MOSEK and SCIP were not able to determine the optimality of solutions within the pre-set time limit of 200 seconds. However, they were able to find the expected solution. Therefore, we measured their calculation time of finding the ground truth solutions from other solvers regardless of the optimality determination.
[c] All measurements of COPT listed in this table are approximate values. Due to the limitation of its interface, we were unable to log the exact time of finding the expected solutions and solution counts.

TABLE V: Comparison of performance for proving infeasibility in problems with 80 VMs. The column *Apps* lists the number of applications in each problem. $\bar{t}$ represents the average calculation time of three random sets in each problem in [s]. σ shows the sample standard deviation of calculation time in [s].

Apps	CPLEX $\bar{t} \pm \sigma$	COPT $\bar{t} \pm \sigma$	Gurobi $\bar{t} \pm \sigma$	MOSEK $\bar{t} \pm \sigma$	SCIP $\bar{t} \pm \sigma$
100	0.38±0.11	0.23±0.01	0.36±0.03	0.54±0.06	0.49±0.03
200	1.91±0.03	1.02±0.02	1.00±0.09	5.96±0.01	2.00±0.09
300	7.02±0.42	3.09±0.12	2.64±0.09	23.18±0.93	5.14±0.13
400	21.03±0.35	6.63±0.09	6.29±0.47	59.89±3.21	9.72±0.25
500	42.49±0.82	11.95±0.21	11.43±0.17	120.60±2.84	16.57±0.67
600	85.03±1.47	18.97±0.34	18.05±1.24	- ± -	25.85±0.67
700	145.16±5.52	28.22±0.09	28.75±0.39	- ± -	38.62±0.81
800	- ± -	40.57±0.44	40.31±0.88	- ± -	54.12±0.18

well as CPLEX as the ground truth for our experiments. Unlike CPLEX and Guorbi, through our investigations, we find that the other solvers cannot autonomously conclude whether the feasible solution found is optimal within a specified time (200 seconds). Thus, they can only return a feasible state. We then experimentally found that although a status of *feasible* is returned, the found solutions contain the optimal solution. To evaluate each solver more fairly, we used a semi-automatic approach for all solvers except CPLEX and Gurobi. We manually configured the stop criteria for MOSEK and SCIP by limiting the number of found feasible solutions and measured their computation time. The exception is COPT, which does not provide an interface to limit the number of solutions. Instead, we reduced the maximum optimization time to approximate the number of solutions. Regardless of the judging criteria, COPT, MOSEK, and SCIP find the expected solutions in a reasonably small number of solution iterations. Their optimization performance is similar



(a) optimization (b) solving (c) infeasible

Fig. 4: Comparison the computational time with respect to the searching for a feasible solution, optimal solution, and detecting an infeasible solution on different solvers. Time limit was set to 200 seconds.

to their solving behavior, where MOSEK is slower than the others and cannot find a feasible solution within 200 seconds in the case of 800 apps. The standard deviation of the searching of optimization time is usually more significant than the standard deviation for searching a feasible solution.

C. *Quantitative comparison on the performance of proving infeasibility*

Table V shows the calculation time of each solver for proving the problems infeasible. $\bar{t}$ represents the average calculation time of three random sets in each problem. σ describes the sample standard deviation of calculation time. We can observe that Gurobi and COPT require the shortest calculation time to prove the infeasibility. SCIP and CPLEX can finish most of the problems within 200 seconds. However, MOSEK can not draw any conclusions in problems with more than 600 applications in a limited time. With a grown number of applications, their calculation time increased as well. Regarding the standard deviation of calculation time, the values are kept in a relatively small range, which implies the solvers have stable behavior of proving the infeasibility of randomly generated applications.

D. *Qualitative comparison on different solvers*

In Fig. 4, we visualize the trends in the average solution and optimization times of solvers for problems of different sizes. By experimenting with different numbers of applications, we can observe that the computation time of all solvers increases exponentially with the number of applications. Besides, the search time for the optimal solution is slightly longer than the search time for the feasible solution, as expected. This is because the feasible solution is the one that covers the optimal solution. In terms of time growth trend, the trend is similar. From the perspective of solver performance for solvable problems, Gurobi outperforms in both solving and optimizing in all scenarios. COPT, CPLEX, and SCIP follow closely. For insolvable problems, Gurobi and COPT show the best performance. Among all solvers, Gurobi, COPT, SCIP requires a shorter time to prove an infeasible problem than to search in a solvable problem. However, in CPLEX and MOSEK, the calculation time of infeasibility analysis is longer than the solving time.

VII. Conclusion

In this paper, we presented an ILP formulation to address the resource allocation problem in SDVs. In such problems, the minimal numbers of VMs should be determined and be allocated with essential resources to host applications with various constraints. We proved the feasibility of the proposed method by utilizing solvers to solve and optimize predefined problems with a reality-based hardware platform and random application sets. Furthermore, we evaluated the abilities of CPLEX, COPT, Gurobi, MOSEK, and SCIP regarding the proposed formulation and analyzed their performance with a scaling number of applications. We found that the solving and optimization time increased exponentially with growing application numbers. Despite performance differences, all solvers can be utilized to solve the resource allocation problem in SDVs automatically. For searching of the optimum solutions with minimal numbers of VMs, both Gurobi and CPLEX are able to automatically find an optimal solution within the given time limits. COPT, MOSEK and SCIP can be integrated into a semi-automated procedure, where human integrators can decide the optimality of solutions or where suboptimal solutions are acceptable.

As a future work, we plan to investigate different formulation strategies, e.g., SMT, and compare with the proposed ILP formulation. Besides, other constraints and aspects such as scheduling ability should also be included to extend the proposed strategy. In addition, heuristic methods will be taken into consideration with a hope of decreasing the solving complexity.

VIII. Acknowledgement

This work was conducted as part of the Huawei-TUM Innovation Lab, sponsored by Huawei Technologies Düsseldorf GmbH.

REFERENCES

[1] Deloitte, "Software-defined vehicles – a forthcoming industrial evolution," https://www2.deloitte.com/cn/en/pages/consumer-business/articles/software-defined-cars-industrial-revolution-on-the-arrow.html, accessed: 2022-02-13.

[2] S. Apostu, O. Burkacky, J. Deichmann, and G. Doll, "Automotive software and electrical/electronic architecture: Implications for oems," https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/automotive-software-and-electrical-electronic-architecture-implications-for-oems, accessed: 2022-02-13.

[3] W. Haas and P. Langjahr, "Cross-domain vehicle control units in modern E/E architectures," in *16. Internationales Stuttgarter Symposium*, Jan. 2016, pp. 1619–1627.

[4] "Road vehicles - functional safety - part 1–10," International Organization for Standardization, Standard, 2011.

[5] D. Reinhardt and G. Morgan, "An embedded hypervisor for safety-relevant automotive E/E-systems," in *Proceedings of the IEEE International Symposium on Industrial Embedded Systems*, 2014, pp. 189–198.

[6] AUTOSAR, "Adaptive platform," https://www.autosar.org/standards/adaptive-platform/, accessed: 2022-02-14.

[7] M. Z. Bjelica and Z. Lukac, "Central vehicle computer design: Software taking over," *IEEE Consumer Electronics Magazine*, vol. 8, no. 6, pp. 84–90, 2019.

[8] P. Hansen, "On automotive electronics," *ATZ Elektron*, vol. 16, pp. 32–34, Nov. 2021.

[9] S. Usorac and B. Pavkovic, "Linux container solution for running android applications on an automotive platform," in *Proceedings of the Zooming Innovation in Consumer Technologies Conference*, 2021, pp. 209–213.

[10] D. Reinhardt, D. Kaule, and M. Kucera, "Achieving a scalable E/E-architecture using AUTOSAR and virtualization," *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 6, pp. 489–497, 2013.

[11] N. Katoh and T. Ibaraki, "Resource allocation problems," in *Handbook of Combinatorial Optimization: Volume 1–3*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1998, pp. 905–1006.

[12] X. Zhang, L. Feng, D.-J. Chen, and M. Törngren, "Design-space reduction for architectural optimization of automotive embedded systems," in *Proceedings of the IEEE International Conference on High Performance Computing and Communications, IEEE International Symposium on Cyberspace Safety and Security, and IEEE International Conference on Embedded Software and Systems*, 2015, pp. 1103–1109.

[13] S. Voss and B. Schätz, "Deployment and scheduling synthesis for mixed-critical shared-memory applications," in *Proceedings of the IEEE International Conference and Workshops on Engineering of Computer Based Systems*, 2013, pp. 100–109.

[14] F. Maticu, P. Pop, C. Axbrink, and M. M. Islam, "Automatic functionality assignment to AUTOSAR multicore distributed architectures," 2016.

[15] S. Martello and P. Toth, "Bin-packing problem," in *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley & Sons, Inc., 1990.

[16] A. Mehiaoui, E. Wozniak, S. Tucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, and L. Lemarchand, "A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems," vol. 48, no. 5, May 2013.

[17] U. Pohlmann and M. Hüwe, "Model-driven allocation engineering: specifying and solving constraints based on the example of automotive systems," *Automated Software Engineering*, vol. 26, pp. 315–378, 2018.

[18] A. Fatima, N. Javaid, T. Sultana, M. Hussain, M. Bilal, S. Shabbir, Y. Asim, M. Akbar, and M. Ilahi, "Virtual machine placement via bin packing in cloud data centers," *Electronics*, vol. 7, no. 12, Dec. 2018.

[19] M. A. Kaaouache and S. Bouamama, "Solving bin packing problem with a hybrid genetic algorithm for VM placement in cloud," *Procedia Computer Science*, vol. 60, pp. 1061–1069, 2015.

[20] H. D. Mittelmann, "Decision tree for optimization software," http://plato.asu.edu/guide.html, accessed: 2022-02-13.

[21] "IBM ILOG CPLEX optimizer," https://www.ibm.com/analytics/cplex-optimizer, accessed: 2022-02-15.

[22] CardinalOperations, "COPT cardinal optimizer," https://www.shanshu.ai/copt, accessed: 2022-02-13.

[23] "Gurobi optimizer," https://www.gurobi.com/products/gurobi-optimizer/, accessed: 2022-02-15.

[24] "MOSEK fusion API for Python 9.3.14," https://docs.mosek.com/latest/pythonapi/index.html, accessed: 2022-02-17.

[25] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig, "The SCIP optimization suite 8.0," Zuse Institute Berlin, ZIB-Report 21-41, Dec. 2021.

[26] S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano, "PySCIPOpt: Mathematical programming in Python with the SCIP optimization suite," in *Mathematical Software – ICMS 2016*. Springer International Publishing, 2016, pp. 301–307.

[27] "Development platform for SOAFEE," https://www.adlinktech.com/en/soafee, accessed: 2022-02-17.