Resource-aware Design for Reliable Autonomous Applications with Multiple Periods*

Rongjie Yan[1], Di Zhu[3,4], Fan Zhang[1,2],
Yiqi Lv[1,2], Junjie Yang[3,4], and Kai Huang✉[3,4]
{yrj,zhangf,lvyq}@ios.ac.cn
{{zhud5,yangjj27}@mail2,huangk36@mail}.sysu.edu.cn

[1] State Key Laboratory of Computer Science, ISCAS, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China
[4] School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China

Abstract. Reliability is the most important design issue for current autonomous vehicles. How to guarantee reliability and reduce hardware cost is key for the design of such complex control systems intertwined with scenario-related multi-period timing behaviors. The paper presents a reliability and resource-aware design framework for embedded implementation of such autonomous applications, where each scenario may have its own timing constraints. The constraints are formalized with the consideration of different redundancy based fault-tolerant techniques and software to hardware allocation choices, which capture the static and various causality relations of such systems. Both exact and heuristic-based methods have been implemented to derive the lower bound of hardware usage, in terms of processor, for the given reliability requirement. The case study on a realistic autonomous vehicle controller demonstrates the effectiveness and feasibility of the framework.

1 Introduction

As the automotive industry is striving for autonomous vehicles through intensive sensing, computation, and communication, a larger number of more complex control applications with guaranteed performances are expected to be on board. Such complex control applications are usually composed of a set of functions that are characterized by various timing behaviors, e.g., environment constraints, sensing/acting frequencies, or various worst case execution times of software components. For these kinds of on-board control applications, reliability is the most critical design issue, as any failure will incur catastrophes. Since

* This work has been partly funded by the National Key Basic Research (973) Program of China under Grant No. 2014CB340701, Key Research Program of Frontier Sciences, CAS, under Grant No. QYZDJ-SSW-JSC036, the CAS-INRIA major project under No.GJHZ1844, the National Science Foundation of China under Grant No. U1435220, No. U1711265, and the Fundamental Research Funds for the Central Universities under grant No.17lgjc40.

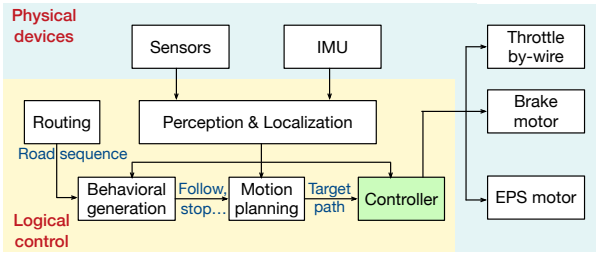


Fig. 1: Functionality of an autonomous controller

algorithm development for these control applications is well-established, system-level mechanisms are mandatory to mitigate the impact of transient faults to ensure system reliability, even though hardware becomes more reliable.

Reliability, however, comes with costs. In principle, system-level mechanisms always adopt active or passive redundancy based fault-tolerant techniques [10]. We consider active redundancy as the major technique to guarantee system reliability, which replicates software tasks into multiple copies. Those copies can be executed on the same processor (temporal redundancy), or distributed to multiple processors (spatial redundancy). In the case of temporal redundancy, additional latency will be introduced which may hamper the response time of the applications. In the case of spatial redundancy techniques, additional hardware is needed to accommodate the replicas. The additionally imposed hardware has to be minimal as automotive industry is particularly sensitive in terms of hardware costs [13]. Therefore, safety-critical control components in automobiles have to be carefully designed, to deploy control components and their redundancy into a given hardware architecture by considering all the constraints.

To motivate our work, let us consider an autonomous automobile controller shown in Fig. 1, whose role is to extract the target path derived from motion planning according to collected information from physical devices, and to control low level actuators to track the path. The frequency of path tracking is usually higher to follow environment updating. Meanwhile, the frequencies of every functionality in different scenarios, such as going straight and making u-turns, are also different. Since, for example, making a u-turn is generally harder than going straight, a higher frequency is required to minimize the tracking error. We expect to adopt minimal number of homogeneous processors for its embedded implementation, to meet reliability guarantees and all timing constraints.

To guarantee reliability and reduce hardware cost at the same time is not easy for such complex control systems intertwined with scenario-related multi-period timing behaviors as well as fault-tolerant mechanisms. The reason is multi-fold. First, scenario-related multi-period timing behaviors incur more design and implementation considerations: 1) timing constraints are scenario-related, and the implementation should accommodate all scenarios; 2) various data dependencies exist, due to the communication between tasks with different periods. Second, hardware optimization for such design is not straightforward: 1) different scenario-related constraints lead to different optimization results; 2) the goal of processor minimization cannot be formatted as an expression that can be cal-

culated with a set of variables and constraints, because it is regarded as a fixed parameter to encode the constraints in the embedded implementation.

To deal with the first challenge, we adopt *hyper period* (the least common multiple of all periods) [14] to unify the scheduling length (makespan) for software to hardware deployment consideration. Meanwhile, we adopt data refreshing technique for communication between tasks with different periods to avoid accessing to empty buffer, where buffered data will not be removed until new data comes and overwrites the old. For the second challenge, to reduce the cost for hardware and the latency for fault tolerance, as well as guaranteeing reliability, we introduce both spatial and temporal redundancy of tasks. Majority fault-free voters are applied to choose the result in majority from multiple replicas, to simplify the implementation. To calculate the least number of required processors, the hardware optimization problem is translated into a satisfiability problem [3], i.e., whether a scheduler satisfying all the constraints exists, with the given number of processors. Then we could provide the result by repeatedly checking the satisfiability problem with various numbers of processors. The method can be employed in various scenarios, and we take the maximum number among the minimized number of processors in various scenarios, such that the implementation is capable of serving all scenarios.

The contributions of the paper are as follows. First, we provide a framework for reliable and resource-aware design of autonomous applications with scenario-related multi-period behaviors, where reducing processor usage is the basis of other resource optimization. Second, we propose an effective method for processor optimization to derive the solution for scenario-related applications. Meanwhile, we present a rule to infer processor usage among various scenarios. We also employ various techniques for solution calculation, such as model checking, constraint solving and heuristic-based methods. The case study on realistic autonomous automobile control systems has demonstrated the feasibility and the effectiveness of our method.

The organization of paper is as follows. Sec. 2 discusses the related work. Sec. 3 concretizes the motivating example. We present the related concepts in Sec. 4. Sec. 5 formalizes the constraints for scenario-related multi-period behaviors and fault-tolerant techniques, and proposes the detailed implementation. Sec. 6 provides the experimental results on the case study, and Sec. 7 concludes.

2 Related work

Automotives are classical instances of mixed-critical systems [5]. We concentrate on reliability and resource-aware design for safety-critical parts of these mixed-critical systems.

Reliability-aware design is widely acknowledged for safety-critical systems [8, 2, 9], which is always regarded as an optimization goal in design space exploration. For example, the work in [9] applies temporal and spatial redundancy and optimize the amount of redundancy with genetic algorithms. For automotive systems, a model based strategy is introduced for soft error tolerance techniques under real-time constraints [16]. The work in [12] analyzes transient errors for automotive safety-critical applications. We consider transient faults caused by

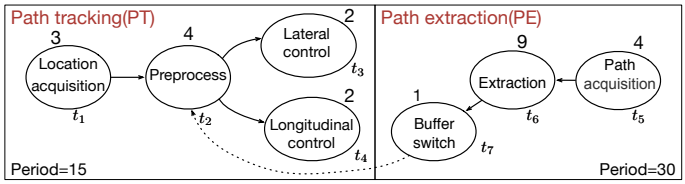


Fig. 2: Task graph of the autonomous controller.

hardware and regard reliability as the fundamental requirement. Once the reliability can be guaranteed, we try to minimize hardware resources, which is a hot topic for embedded systems [6, 15]. The work in [6] considers the optimization of hardware resources for multi-rate automotive control systems on single-processor platforms. Zhao et al. concentrates on stack usage minimization for AUTOSAR models [15]. Our concern is the minimization of processors, which is also the basis of other resource optimizations.

To deal with data communication in multi-period systems, various techniques, such as communication protocols [11] and *lossless buffering* [14], have been introduced. In [14], base period (the *greatest common divisor* of all task periods) is adopted as the length of the scheduling. We apply hyper period of all tasks, and the communication is implemented with data refreshing technique.

3 The Motivating example

We present in Fig. 2 the structure of the controller[5] mentioned in Sec. 1, whose role is to receive a target path and control low level actuators to track the path. It consists of two components: 1) Path tracking (PT), to track the path according to the input from IMU (Inertial Measurement Unit); 2) Path extraction (PE), to process the target path calculated from a motion planning module.

PT consists of three processes: location acquisition, preprocessing and control instruction output. Once data from IMU is available (t_1), tracking error will be calculated in the preprocessing step (t_2). This step also considers the output from the buffer switch task (t_7) in PE when it is available. Next, lateral control (t_3) and longitudinal control (t_4) run in parallel. In the former, steering angle is calculated by a structure with both feedforward controller and feedback controller, and supplemented by yaw damping. The latter includes two PID controllers for throttle and brake, respectively.

PE involves three steps: path acquisition (t_5), extraction (t_6) and buffer switch (t_7). Once a path is acquired in the first step, it will be delivered to the extraction step. The extraction step mainly targets for spline interpolation, radius calculation and other relevant computations. After the extraction, the double buffer implemented for parallel writing and reading will be updated.

Difference in sampling rates makes the periods of PT and PE different. And the periods of PT (or PE) are different in various scenarios, such as making a u-turn, or going straight, though the worst case execution time (WCET) of every task in all scenarios is the same.

[5] To ease the description, periods, computation costs (labelled on tasks) presented here are simplified.

4 Preliminaries

To globally consider multi-period timing behaviors among various subsystems in a scenario and optimize hardware resources, we first propose the concept of *atypical task graph*. Then we present the communication model for multi-period behaviors. Finally, the fault-tolerant techniques applied here are recapped.

4.1 Atypical task graph

Every task can be encoded as a tuple with $t = (id, \delta, w)$, where id is the identity of the task, δ indicates the cost of worst case execution, and w shows the degree of importance. A subsystem can be described with an acyclic directed graph $G_i = \langle T_i, E_i \rangle$ and period of occurrence $\mathcal{P}_i$, where T_i is a finite set of tasks, and $E_i \subseteq T_i \times T_i$ is a set of precedence relations with $c_e : E_i \to \mathbb{N}$ to indicate the cost of data transferred between each pair of tasks. A system is composed of a set of subsystems and the connection maintained by data transfer between these subsystems. Due to the difference in periods, a task in a subsystem may ignore the unavailable resource from another subsystem. Consequently, we introduce strong and weak dependency for the relations of tasks in a global system and formalize them in the model of *atypical task graph*.

Definition 1 *An atypical task graph is a tuple* $\mathcal{G} = \langle T, E \rangle$*, where* $T = \bigcup_{i=1}^{n} T_i$*, and* $E \subseteq \{T_i \times T_j \mid 1 \leq i, j \leq n\}$*, with* $G_i = \langle T_i, E_i \rangle \in \mathcal{G}$*. For* $t_i, t_j \in T$*, we have*

- *strong dependency: if* $t_i \to t_j \in E_i$*,* t_j *has to wait for the output of* t_i*,*
- *weak dependency: if* $t_i \rightsquigarrow t_j \in E \setminus (\bigcup_{i=1}^{n} E_i)$*,* t_j *can ignore the output of* t_i*.*

In the model of Fig. 2, we have $t_7 \rightsquigarrow t_2$ (which is connected with dashed line). Other precedence relations are strong.

4.2 Communication model

Communication for weak dependency relation is implemented with buffer refresh semantics, i.e., the sink task reads the old data in the buffer until it is refreshed. Given two tasks t_i and t_j from two subsystems with periods $\mathcal{P}_i$ and $\mathcal{P}_j$, respectively, if $t_j \rightsquigarrow t_i$, their communication scenario can be described as the case in Fig. 3, where the arrow shows the direction of data transfer for the weak dependency relation. In the scheduling of Fig. 3, though t_j finishes its execution at time point σ_1, t_i has to wait for additional σ_2 time units to use the refreshed data. Intuitively, the number of iterations for t_i to obtain the refreshed data is in the range of $[\lceil \sigma_1/\mathcal{P}_i \rceil, \lceil \mathcal{P}_j/\mathcal{P}_i \rceil + 1]$, where in the worst case t_j is scheduled in the end of its subsystem.

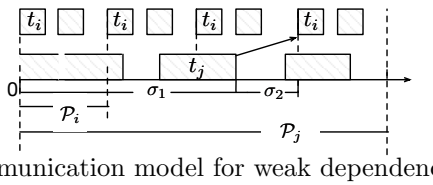
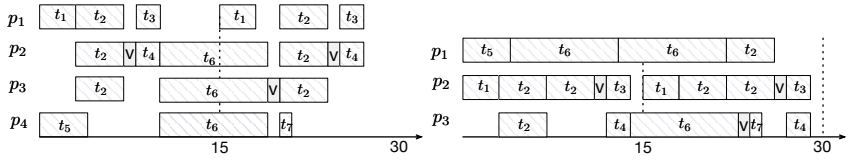


Fig. 3: Communication model for weak dependency relation.

For safety-critical systems, we may expect to reduce the time that one has to wait for the other, i.e., minimize σ_1 in the scheduling of Fig. 3. The optimization is a local scheduling for the corresponding subsystem.

4.3 Active redundancy based fault tolerance

We consider spatial redundancy, and spatial and temporal mixed redundancy in the paper. Intuitively, temporal redundancy will prolong the execution of tasks, and spatial redundancy will require more hardware resources. The mixture of the two may reduce these disadvantages. Consider the model given in Fig. 2. When three replicas exist for t_2 and t_6, respectively, we present two schedulers in Fig. 4 with two redundancy strategies, where applying pure temporal redundancy violates timing constraints and is ignored. In the case of spatial redundancy shown in Fig. 4(a), four processors are required to satisfy timing requirements. However, the case of mixed redundancy in Fig. 4(b) only needs three processors, though data update for t_2 from t_7 is delayed by one period of PT.



(a) Spatial redundancy (b) Mixed redundancy

Fig. 4: Two schedulers for the model in Fig. 2.

5 Reliable and resource-aware design

Two design objectives, i.e., reliability guarantee and processor minimization, introduce two optimization steps: 1) calculate the redundancy degree of every task for system reliability, by assuming that communication between tasks is reliable; 2) find the minimal number of processors such that all the constraints can be satisfied. We first present restrictions on satisfying reliability requirements, and constraints on adopting various fault-tolerant techniques with multi-period timing behaviors. Then we discuss the implementation for optimizing two goals.

5.1 Redundancy optimization

We adopt the Poisson fault model [2] to compute the success/failure probability of tasks. Given task t_i and processor p with failure rate λ_p, the probability for t_i executing correctly on processor p is $\mathscr{P}_i = e^{-\lambda_p \delta_i}$. Then the probability of t_i encountering a transient fault is $1 - \mathscr{P}_i$. We employ $\mathscr{P}_i$ as the reliability of t_i.

Given a system depicted with an atypical task graph $\mathcal{G}$, we first evaluate the reliability of its subsystems. For the task graph G_k of a subsystem, if one of the tasks fails, it is not reliable. Therefore, for the task graph with $|T_k|$ tasks, where the reliability of every task is $\mathscr{P}_i$, the reliability of its subsystem is

$$R_k = \prod_{i=1}^{|T_k|} \mathscr{P}_i. \tag{1}$$

If a task contains r replicas, its reliability becomes $1 - (1 - \mathscr{P}_i)^r$, which is greater or equal to $\mathscr{P}_i$. As redundancy can increase reliability, the subsystem

reliability can be enhanced with replicas of its tasks. Let r_i be the number of replicas for task t_i with reliability $\mathscr{P}_i$, we have

$$R_k = \prod_{i=1}^{|T_k|}(1-(1-\mathscr{P}_i)^{r_i}). \tag{2}$$

Given a requirement that the system reliability should be at least $\mathcal{R}$, we can calculate the minimal number of replicas for all the tasks in a system, such that all the subsystems satisfy the reliability requirement, i.e.,

$$\begin{aligned} &minimize(\textstyle\sum_{T_k \subseteq T} \sum_{i=1}^{|T_k|} r_i \cdot w_i) \\ &\text{subject to:} \\ &\qquad \text{forall } k,\ \textstyle\prod_{i=1}^{|T_k|}(1-(1-\mathscr{P}_i)^{r_i}) \geq \mathcal{R} \end{aligned} \tag{3}$$

where w_i is the weight of task t_i, and $|T_k|$ is the number of tasks in T_k.

We consider a majority voter, to generate an output if and only if more than half of the inputs have the same value. And the reliability of a voter is assumed to be 1. A voter can be regarded as a task by inserting it into the task graph, according to the dependency relation of its predecessor.

5.2 Constraint formalization and resource optimization

Table 1: Constraints and variables

Const.	Explanation	Var.	Explanation
N_i	the number of iterations for task t_i	o_{ij}	indicating the existence of communication
r_i	the number of replicas for task t_i	m^u_{ijk}	the jth replica of task t_i is mapped to p_k in iteration u
$\mathcal{P}_i$	the period of task t_i	s^u_{il}	start time of executing the lth replica of t_i in iteration u
δ_i	the cost of executing task t_i	f^u_{il}	end time of executing the lth replica of t_i in iteration u
d_{ij}	dependency relation for t_i and t_j	se^u_{ij}	time for starting data transfer from t_i to t_j in iteration u
c_{ij}	cost of data transfer from t_i to t_j	fe^u_{ij}	time for finishing data transfer from t_i to t_j in iteration u
		a^u_i	arrival time for t_i in iteration u

To formalize the mapping and scheduling constraints for the corresponding embedded implementation, we assume that the number of replicas for every task has been calculated, and all the necessary voters are converted into tasks. The necessary notations for constants and variables are listed in Table 1. For multiple periods, we first compute the least common multiple M of these periods. Then the number of iterations of every task in a hyper period is $N_i = M/\mathcal{P}_i$. We introduce d_{ij} to record the precedence relation between pairs of tasks, where $d_{ij} = 0$ stands for the non-existence of dependency relation, $d_{ij} = 1$ is for the strong dependency relation, and $d_{ij} = -1$ is for the weak dependency relation.

Mapping and scheduling constraints The constraints presented here mainly involve the mapping of tasks and replicas, the behaviors between strong and weak dependent tasks, and the causality between various actions. For the type of redundancy, we consider the cases of spatial, spatial and temporal mixed redundancy. For the mapping relation between tasks and processors, it can be

Table 2: Case-specified mapping constraints

Type	Fixed mapping	Flexible mapping
Spatial	$\sum_{j=1}^{r_i}\sum_{k=1}^{\lvert P\rvert} m_{ijk} = r_i, \quad \sum_{j=1}^{r_i} m_{ijk} \leq 1$ (4)	$\sum_{j=1}^{r_i}\sum_{k=1}^{\lvert P\rvert} m^u_{ijk} = r_i, \quad \sum_{j=1}^{r_i} m^u_{ijk} \leq 1$ (5)
Mixed	$\sum_{j=1}^{r_i}\sum_{k=1}^{\lvert P\rvert} m_{ijk} = r_i$ (6)	$\sum_{j=1}^{r_i}\sum_{k=1}^{\lvert P\rvert} m^u_{ijk} = r_i$ (7)

fixed (the mapping will not change in various iterations) or flexible (the mapping can change among various iterations).

The constraints in Table 2 depict the mapping restriction in various cases. For spatial redundancy, every processor $p \in P$ can only accommodate one replica of a task. However, in the mixed case, such limitation does not exist. If the mapping is fixed, the allocation relations of tasks to processors keep the same in all the iterations, and we ignore the iteration index.

Equation 8 requires that in spatial redundancy, all replicas of a task should be executed at the same time, which is not involved in mixed redundancy.

$$\forall 1 \leq l, l' \leq r_i, s^u_{il} = s^u_{il'} \tag{8}$$

The general causality constraints on scheduling are depicted in Table 3.

Table 3: General constraints

Explanation	Constraints
For any two dependent tasks, if they are not allocated to the same processor, communication exists.	$(d_{ij} \neq 0) \wedge (\exists k, k'.(m^u_{ilk} \wedge m^u_{jl'k'})) \rightarrow o_{ij}$ (9)
The quantitive relation exists between task execution and data transformation.	$f^u_{il} = s^u_{il} + \delta_i,\ fe^u_{ij} \geq se^u_{ij} + o_{ij} \cdot c_{ij}$ (10)
Causality exists between data transfer and subsequent tasks.	$se^u_{ij} \geq f^u_{il},\ (d_{ij} = 1) \rightarrow s^u_{jl} \geq fe^u_{ij}$ (11)
Tasks executing on the same processor cannot overlap.	$m^u_{ilk} \wedge m^v_{jl'k} \rightarrow s^u_{il} \geq f^v_{jl'} \vee s^v_{jl'} \geq f^u_{il}$ (12)
The arrival of tasks is periodic.	$\alpha^{u+1}_i - \alpha^u_i = \mathcal{P}_i \wedge s^u_{il} \geq \alpha^u_i \wedge u + 1 \leq N_i$ (13)

Objectives The reliability requirement demands sufficient number of processors to accommodate redundancy and to satisfy the timing requirements. Meanwhile, we also expect to reduce the adopted hardware resources for cost consideration. Therefore, we expect to minimize the number of allocated processors without sacrificing system reliability.

Let T be a set of tasks with $|T| = n$, S be a set of scenarios of a system with $|S| = m$, and $\mathcal{P}_{ij}$ be the period of task t_i in scenario s_j. The minimum number of processors we need is

$$max\{minimize(|P_j|) \mid 1 \leq j \leq m\} \tag{14}$$

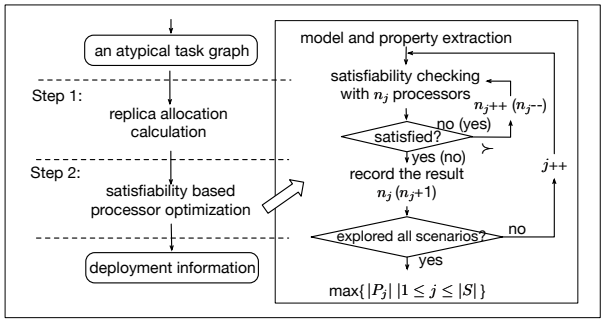


Fig. 5: Optimization steps

where $minimize(|P_j|)$ is the minimum number of processors used in scenario s_j to satisfy time constraints in the hyper period by considering the set of periods $\{\mathcal{P}_{ij}\}_{1\leq i\leq n}$. To satisfying the constraints of all the scenarios, we need to select the maximum among all the results.

Theorem 1. *Given a system with a set of tasks T and a set of applied scenarios S, where $\mathcal{P}_{ij}$ is the period of task $t_i \in T$ in scenario $s_j \in S$, let M_j be the least common multiple of all tasks T in scenario s_j, and $|P_j|$ be the minimal number of required processors for a satisfiable scheduler in scenario s_j. If there exists $s_{j'} \in S$ such that*

$$(M_j \leq M_{j'}) \wedge \bigwedge_{i=1}^{|T|}(\frac{M_j}{\mathcal{P}_{ij}} \geq \frac{M_{j'}}{\mathcal{P}_{ij'}}),$$

the tasks in s'_j is schedulable with $|P_j|$ processors.

Proof. If $|P_j|$ is the number of required processors for scenario s_j, we have

$$\sum_{i=1}^{|T|} \frac{M_j}{\mathcal{P}_{ij}} \cdot \delta_i \leq M_j \cdot |P_j|.$$

Then we have

$$\sum_{i=1}^{|T|} \frac{M_{j'}}{\mathcal{P}_{ij'}} \cdot \delta_i \leq \sum_{i=1}^{|T|} \frac{M_j}{\mathcal{P}_{ij}} \cdot \delta_i \leq M_j \cdot |P_j| \leq M_{j'} \cdot |P_j|.$$

Therefore, the tasks in $s_{j'}$ is schedulable with $|P_j|$ processors.

Informally speaking, with Theorem 1, we could save the effort of optimization by ignoring the scenarios that the hardware resource usage can be inferred. The reason is that we need to satisfy the requirements of all scenarios.

5.3 Implementation

We adopt a stepwise strategy for the optimization of the goals, as shown in Fig. 5. First, we introduce a greedy algorithm to calculate the allocation of replicas, where every subsystem should satisfy the reliability requirement. Then, the optimization for the minimal number of processors is translated into a satisfiability problem, such that the existence of a deployment strategy satisfying all the constraints can be checked.

Replica calculation Increasing the number of replicas for tasks with lower reliabilities is more effective in enhancing system reliability. Therefore, the greedy algorithm tends to assign more replicas to such tasks. Meanwhile, the number of replicas is set to be odd for majority voting. If all the tasks have the same weight, the calculated configuration of replicas is the optimized solution. However, when the weights are different, there may exist many solutions for a given expected system reliability. The algorithm just outputs one replica configuration.

Hardware resource minimization It is difficult to directly apply constraint-based or meta-heuristic based optimization techniques to minimize the number of adopted processors. The reason is that an optimization objective is usually encoded as an expression that can be calculated with a set of variables and constraints. However, the number of processors is regarded as a fixed parameter to encode the constraints in the design for an embedded implementation. The intuition of the problem is to find a minimal number of processors, such that the system is schedulable with the constraints. When the number is given, checking whether there exists a scheduler meeting the constraints is a satisfiability problem. Then we can use various techniques, such as model checking, constraint solving or heuristic-based methods, for satisfiability analysis. Therefore, we could keep on checking the satisfiability of the constraints mentioned in Sec. 5.2 with various numbers of processors, until we reach the minimum such that all the constraints in a certain scenario are satisfied.

We have encoded the constraints in various cases (the product between mapping and redundancy choices) with model checking, constraint solving and heuristic-based methods. As illustrated in the right-side of Fig. 5, the method implemented with model checking works as follows:

1. we build a formal model to depict the constraints for tasks (replicas) being executed on processors, such that the minimal number of required processors to satisfy the constraints in the model can be checked.
2. the property we check is whether all the tasks can be done within a specified deadline (hyper period) of scenario s_j.
3. for a given number of processors n_j,
   - if the property is satisfied, reduce n_j by one and check the satisfiability of the property until it is not satisfied. Then $n_j + 1$ is the result in s_j.
   - if the property is not satisfied, increase n_j by one and check the satisfiability of the property until it is satisfied. Then n_j is the result.

The condition marked with $\succ$ in Fig. 5 must be explored at least once. Steps 2 and 3 are repeated until all necessary scenarios have been checked. Then the maximal number among all the scenarios is the expected result.

The model mainly contains templates for tasks and processors, which are formalized with timed automata [1]. The execution of tasks on processors is encoded as the coordination between the corresponding components.

We provide the templates for flexible mapping and mixed redundancy in Fig. 6 (a) and (b) with two timed automata for tasks (replicas) and processors, respectively, where the nodes with double cycles are initial locations. The condition labelled on edges between two locations describes the constraints for the

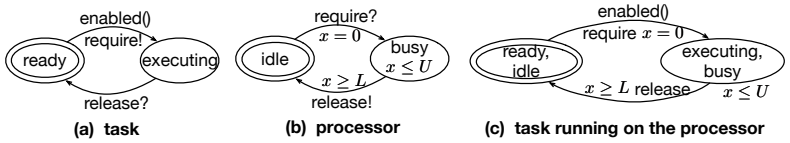


Fig. 6: Models in timed automata

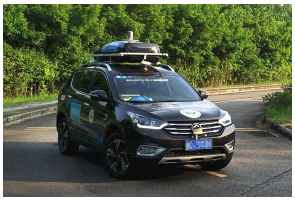
transition. There is a clock variable x in the processor model. When a task starts execution on a processor, x is reset to zero to record time elapse. When x reaches L, the task can release the occupation of the processor. The constraint $x \leq U$ requires that the execution of a task should not exceed U (the WCET of a task). The two timed automata coordinate via the message marked with ! and ?. The composition of task and processor via message synchronization is shown in Fig. 6(c). Additional to the constraint of deadline, we could also encode the timing requirements between weak dependent tasks that the scheduler should satisfy. The templates can be instantiated with multiple tasks and processors. Then we can check whether a scheduler satisfying all the timing constraints exists.

We employ model checker UPPAAL [4] to deal with the checking process. Once we fix the minimal number of processors, a counter-example showing the configuration and scheduling strategy can be given. If the given number of processors cannot guarantee to satisfy all the constraints, the whole state space will be explored by using model checking techniques. Then the model checker may not present an answer due to state space explosion for large scale systems. With this consideration, we also apply SMT solver Yices [7] to encode and solve the constraints. Additionally, a heuristic-based method is implemented with some greedy strategies, which is sound but not complete. That is, if the algorithm can find a scheduler, adopting the given number of processors can satisfy the constraints. However, if the algorithm fails to find a scheduler within finite number of iterations, we cannot say that the system is not schedulable with the given number of processors.

6 Case study

We conduct experiments on the autonomous vehicle controller depicted in Fig. 1. The autonomous vehicle is modified from Dongfeng Fengshen AX7 SUV (Fig. 7(a)) with drive-by-wire ability. The vehicle is equipped with a variety of sensors and low-level actuators. The actuators, e.g., electronic power steering motor, brake motor, and throttle by-wire, receive and actuate commands from the controller. Vehicle data such as current steering angle is sensed by on-board sensors and obtained through a CAN bus. The IMU used for localization is an Inertial Navigation System (INS) aided by external Differential Global Positioning System (DGPS). Other sensors, e.g., LiDAR, Radar and camera, provide data for the centre computer. Then the centre computer delivers environment perception and generates a target path that is a sequence of position points with maximum speed information. Next, the controller keeps the vehicle tracking this path.

In the autonomous control system (Fig. 7(b)), the controller is a real-time application running under a Linux kernel with PREEMPT_RT patch. The type


(a) Dongfeng Fengshen AX7


(b) Autonomous control systems

Fig. 7: Our autonomous vehicle

of processors for controller execution is Raspberry Pi 3, namely a 1.2GHz quad-core ARM Cortex A53 cluster. The communication between various processors is via Ethernet. We conduct real urban road tests as well as simulation tests with scenarios covering straight, curve, lane change, and u-turns. The frequencies of PE and PT among various scenarios of the vehicle are listed in Table 4, where "NA" means not available. The execution times of the tasks are recorded when the controller runs on one processor. Their worst case execution times are analyzed based on the collected data, as presented in Table 5.

Table 4: Various frequencies (Hz) of PE and PT in different scenarios

Speed	Path tracking (PT)			Path extraction (PE)		
	Straight	Turn	U-turn	Straight	Turn	U-turn
10km/h	100	100	120	10	10	12
20km/h	100	100	181	10	10	19
30km/h	100	107	NA	10	11	NA
40km/h	120	NA	NA	12	NA	NA
60km/h	197	NA	NA	20	NA	NA

Table 5: Worst case execution time of the tasks

Type	Path tracking (PT)				Path extraction (PE)		
tasks	t_1	t_2	t_3	t_4	t_5	t_6	t_7
WCET(ms)	0.2842	0.0820	0.6674	0.5932	0.1306	6.7042	0.0868

6.1 Redundancy degree for various reliability requirements

The numbers of required replicas with respect to various system reliabilities, the reliabilities of tasks and the weights of the tasks are listed in Table 6. Totally, we consider four tasks in PT and present two groups of weights $w_1 = (0.25, 0.25, 0.25, 0.25)$, and $w_2 = (0.33, 0.01, 0.33, 0.33)$[6], and four groups of task reliabilities, to compare the number of required replicas with various system reliability requirements. The four groups of task reliability distributions in ascending order are:

$D_1 = (0.96, 0.99, 0.94, 0.94)$, $D_2 = (0.996, 0.999, 0.994, 0.994)$

$D_3 = (0.9996, 0.9999, 0.9994, 0.9994)$, $D_4 = (0.99996, 0.99999, 0.99994, 0.99994)$

The legends in Table 6 are as follows. The first column lists various reliability requirements. The second column presents different weights explained above. The

[6] The smaller the value is, the more important the task is.

other four columns provide the number of required replicas for the four tasks in PT, respectively, with respect to various distributions of reliabilities and weights of the tasks.

Table 6: The number of replicas for various reliability requirements

Reliability	Weight	D_1	D_2	D_3	D_4
$1-10^{-6}$	w_1	(5,5,7,5)	(3,3,3,3)	(3,3,3,3)	(3,3,3,3)
	w_2	(5,5,7,5)	(3,3,3,3)	(3,3,3,3)	(3,3,3,3)
$1-10^{-9}$	w_1	(7,5,9,9)	(5,5,5,5)	(3,3,3,3)	(3,3,3,3)
	w_2	(7,7,9,9)	(5,5,5,5)	(3,3,3,3)	(3,3,3,3)
$1-10^{-12}$	w_1	(9,7,11,11)	(7,5,7,7)	(5,5,5,5)	(3,3,3,3)
	w_2	(9,7,11,11)	(7,5,7,7)	(5,5,5,5)	(3,3,3,3)

According to the results presented in Table 6, when the reliabilities of tasks are lower, more replicas are required to meet system reliability requirements. However, the allocation of replicas is less sensitive to weights, except for the case with the lowest task reliabilities.

The total numbers of replicas in various reliability requirements and weights with respect to more distributions of task reliabilities are illustrated in Fig. 8 (i in D_i stands for the degree of reliabilities, i.e., the bigger the number is, the more reliable the tasks are). In every reliability requirement, the numbers of required replicas in two sets of weights are similar, except for the case already shown in Table 6. When task reliabilities are higher, the change in the number of required replicas is not so obvious as the change in reliability requirements.

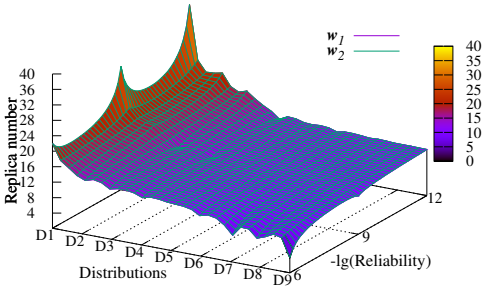


Fig. 8: Replicas distribution.

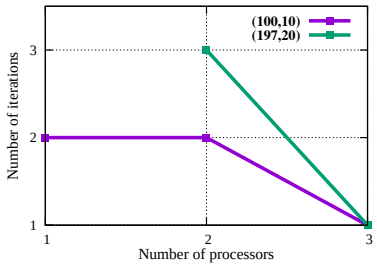


Fig. 9: PT's iteration for fresh data.

6.2 Resource optimization within various scenarios

Hardware resource is sensitive to the number of replicas and the corresponding timing requirements. We take two sets of experimentation to compare the usage of hardware resource with respect to various scenarios, and different fault-tolerant strategies, i.e., spatial redundancy, spatial and temporal mixed redundancy, respectively. The experiments are conducted with the three methods mentioned in Sec. 5.3, to compare the performance of various techniques in the satisfiability-based optimization.

In Table 7, we present the number of required processors by considering spatial redundancy, with different mapping strategies and various numbers of task

replicas in various scenarios. In the table, the first column presents the frequencies of various scenarios. The second lists the number of replicas for various tasks, where $t_i \times r$ stands for the existence of r replicas for all the tasks. The third presents the number of used processors. Then the rest of columns provide the satisfiability of the problem (the existence of a scheduler), and the cost of the computation in seconds, with constraint solving, model checking and greedy algorithm.

Table 7: Optimization with spatial redundancy

Frequency (Hz)	Replica	$\|P\|$	Constraint solving				Model checking				Greedy algorithm			
			fixed map.		flexible map.		fixed map.		flexible map.		fixed map.		flexible map.	
			sat.	cost	sat.	cost	sat.	cost	sat.	cost	sat.	cost	sat.	cost
(100,10)	$t_i \times 1$	1	Y	0.041	Y	0.037	Y	0.010	Y	0.010	Y	0.003	Y	0.003
	$t_i \times 3$	3	Y	0.161	Y	0.189	Y	7148.820	Y	7218.180	Y	0.003	Y	0.004
(120,12)	$t_i \times 1$	1	Y	0.036	Y	0.035	Y	0.010	Y	0.010	Y	0.003	Y	0.003
	$t_i \times 3$	3	Y	0.095	Y	0.090	Y	5464.94	Y	6714.620	Y	0.003	Y	0.004
(107,11)	$t_i \times 1$	1	Y	0.040	Y	0.039	Y	0.010	Y	0.010	Y	3.579	Y	3.073
	$t_i \times 3$	3	Y	0.098	Y	0.103	Y	7083.540	Y	8520.740	N	32.574	N	32.251
(197, 20)	$t_i \times 1$	1	Y	0.070	Y	0.073	Y	0.020	Y	0.010	Y	0.003	Y	0.004
	$t_i \times 3$	3	Y	0.198	Y	0.189	Y	7183.370	Y	8494.460	N	30.043	N	30.681

Experimental results show that the performance of constraint solving is better than the other two methods. When the total number of replicas is small, the method of model checking may find a solution very quickly. However, with the increasing scalability, the performance of model checking degrades rapidly. For the case of heuristic-based method, the heuristic used here is not always effective, which may fail to find a solution, though it exists. Among the results of fixed mapping and flexible mapping, the model checking cost of the latter is almost always higher than that of the former, due to the increased complexity for the allowed flexibility. For these scenarios, it is enough to use three processors to accommodate the spatial redundancy of three replicas for each task to satisfy reliability requirements.

Table 8: Optimization with spatial and temporal mixed redundancy

Frequency (Hz)	Replica	$\|P\|$	Constraint solving				Model checking				Greedy algorithm			
			fixed map.		flexible map.		fixed map.		flexible map.		fixed map.		flexible map.	
			sat.	cost	sat.	cost	sat.	cost	sat.	cost	sat.	cost	sat.	cost
(100, 10)	$t_i \times 3$	1	Y	189.207	Y	228.648	Y	4.790	Y	2.650	N	106.903	N	110.818
		2	Y	0.472	Y	1.127	Y	0.012	Y	0.013	Y	0.003	Y	0.006
(120, 12)	$t_i \times 3$	1	Y	152.501	Y	150.642	Y	3.510	Y	2.540	N	132.833	N	117.347
		2	Y	1.140	Y	1.235	Y	0.014	Y	0.013	Y	0.004	Y	0.004
(107, 11)	$t_i \times 3$	1	Y	102.439	Y	101.999	Y	3.500	Y	2.560	N	119.581	N	142.081
		2	Y	1.167	Y	1.088	Y	0.011	Y	0.012	Y	0.004	Y	0.004
(197, 20)	$t_i \times 3$	1	N	5708.326	N	4232.964	N	3.180	N	2.370	N	75.561	N	75.964
		2	Y	4.506	Y	3.564	Y	0.014	-	-	Y	0.004	Y	0.003

We present the experimental results with spatial and temporal mixed redundancy in Table 8. In the table, "-" means out of memory. The underlined results in model checking are obtained by applying under approximation to relieve state space explosion, which is sound for satisfiability problem if the result is positive[7].

[7] We can also run the method to check the results in Table 7.

Intuitively, the delay of mixed redundancy is larger than the case with only spatial redundancy, which is a trade-off between time and space. However, for the first three scenarios, only one processor is enough to accommodate three replicas for all the tasks, which is benefited from the small portion of task execution with respect to the periods of the two subsystems. When we allow two processors (the number of solutions is larger), these methods can find a solution more quickly, except for the fourth scenario with the model checking method.

According to the results presented in Tables 7 and 8, in fact we could just calculate the number of required processors for the last scenario, where others can be inferred according to Theorem 1.

6.3 Timing constraint on weak dependency

For the weak dependency between two tasks, we have presented the upper and lower bounds of iterations that the successor should wait in Sec. 4.2. In our case study, we expect that data can be refreshed as early as possible in every hyper period. Therefore, we have encoded the constraints such that the property can be checked. Experimental results show that in the first iteration of PT, data from PE cannot be refreshed with the existing scenarios, for the mixed redundancy with one or two processors. Only using three processors can meet such constraint. In Fig. 9, we present the number of minimal iterations for PT to acquire the updated data from PE with various numbers of processors and various scenarios[8], where the horizontal shows the number of processors, and the vertical stands for the number of iterations. It is obvious that data can be refreshed earlier for PT with more processors. And when the frequencies are lower, it takes fewer iterations to be refreshed.

Concluded from the experimental results, we could adopt three processors in the implementation to accommodate various design considerations.

7 Conclusion

The paper presents an embedded design framework for safety-critical systems with scenario-related multi-period timing behaviors in autonomous vehicles. The main technical challenge is to guarantee system reliability and minimize processor usage, with various timing constraints and design choices. We have formalized the constraints and employed both exact (model checking and constraint solving) and heuristic-based (greedy algorithm) methods for solution calculation. The realistic case study for the controller of an autonomous vehicle has demonstrated the applicability and flexibility of our framework. As the future work, we are interested in considering the reliability issues in mixed-critical systems of autonomous vehicles.

Acknowledgments

The authors would like to thank Jian Zhang and Feifei Ma for their assistance with the work and valuable comments on this paper.

[8] As the cases of (120,12) and (107,11) coincidence with the case of (100,10), we ignore them in the figure.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. P. Axer, M. Sebastian, and R. Ernst. Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In *CODES+ISSS*, pages 149–158. IEEE/ACM/IFIP, 2011.
3. C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking*. MIT press, 2008.
4. G. Behrmann, A. David, and K. Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 33–35, 2004.
5. A. Burns and R. Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.
6. W. Chang, S. Chakraborty, et al. Resource-aware automotive control systems design: A cyber-physical systems approach. *Foundations and Trends® in Electronic Design Automation*, 10(4):249–369, 2016.
7. B. Dutertre. Yices 2.2. In *CAV*, pages 737–744. Springer, 2014.
8. M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich. Reliability-aware system synthesis. In *DATE*, pages 1–6, 2007.
9. J. Huang, S. Barner, A. Raabe, C. Buckl, and A. Knoll. A framework for reliability-aware embedded system design on multiprocessor platforms. *Microprocessors and Microsystems*, 38(6):539–551, 2014.
10. J. Jiang and X. Yu. Fault-tolerant control systems: A comparative study between active and passive approaches. *Annual Reviews in control*, 36(1):60–72, 2012.
11. C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete event dynamic systems*, 21(3):307–338, 2011.
12. S. Pandey and B. Vermeulen. Transient errors resiliency analysis technique for automotive safety critical applications. In *DATE*, page 9, 2014.
13. A. Sangiovanni-Vincentelli and M. Di Natale. Embedded system design for automotive applications. *Computer*, 40(10), 2007.
14. E. Yip, M. M. Kuo, P. S. Roop, and D. Broman. Relaxing the synchronous approach for mixed-criticality systems. In *RTAS*, pages 89–100. IEEE, 2014.
15. Q. Zhao, Z. Gu, and H. Zeng. Design optimization for AUTOSAR models with preemption thresholds and mixed-criticality scheduling. *Journal of Systems Architecture*, 72:61–68, 2017.
16. B. Zheng, H. Liang, Q. Zhu, H. Yu, and C.-W. Lin. Next generation automotive architecture modeling and exploration for autonomous driving. In *VLSI (ISVLSI)*, pages 53–58. IEEE, 2016.