DSI DATA SCIENCE AND INFORMETRICS
VOL.1, NO.2, MAY. 2021

Quantitative analysis of real–time performance and hardware requirements for edge computing platform

Kangkang Shi[a,b], Gangyong Jia [a,b*], Youhuizi Li[a,b,] Yuyu Yin[a,b], Congfeng Jiang[a,b], Li Zhou[a,b]

a. School of Computer Science, Hangzhou Dianzi University, Hangzhou, China
b. Key Laboratory of Complex Systems Modeling and Simulation, Hangzhou Dianzi University, Hangzhou, China

ABSTRACT

For real-time edge systems such as autonomous driving, not only the correctness of task functions, but also the response and processing time of tasks should be satisfied. In the hardware selection phase of a real-time system, time series analyses must be performed on the hardware platform running real-time applications. At present, the common method of worst-case execution time (WCET) analysis focuses mainly on analyzing the impact of hardware platform architecture or task execution process on the task running time. However, different tasks in an autopilot system have different levels of urgency, and preemption between tasks is the main factor that affects the task execution time. The key problem is how to quantify the time fluctuation caused by task preemption for each subtask of the autopilot system running on a fixed hardware platform. This paper presents a time analysis method for a real-time application based on a queuing theory and preemptive scheduling strategy, which assigns different priorities to tasks according to their time urgency and preemptive scheduling according to task priority. Through an experimental case study, the impact of the running time of each subtask in a real-time application with task priority preemptive scheduling is analyzed, along with the impact of changes in hardware platform performance on such real-time applications.

KEYWORDS

Autonomous Driving; Queuing Theory; Preemptive Scheduling; Time Series Analysis.

1 Introduction

With the rise of edge computing, autonomous driving technology has developed rapidly (Yurtsever et al., 2020). Self-driving cars must extract meaningful information from an amount of raw data collected by sensors, understand the surrounding environment, and continuously make decisions based on changes in that environment (Lim et al., 2019). Most of the tasks in this process are urgent, and the most important feature is that data processing and event response have strict time constraints. For example, the completion time for target detection, vehicle obstacle avoidance, target positioning, path planning, and other

*corresponding author: gangyong@hdu.edu.cn

subtasks should be accurately controlled within a safe range. Once the task exceeds the time limit, it may bring unimaginable consequences (Bock et al., 2018; Li et al., 2020). Therefore, the selection of a hardware computing platform that can meet the real-time requirements of an autopilot system must be considered in the design process of the system. Consequently, in the design of an autonomous driving control system, it is necessary to analyze and predict the time fluctuation of each subtask when it is executed periodically on the fixed hardware platform.

The WCET is often used as an indicator to evaluate whether a task can be completed on time (Wilhelm et al., 2020). Mezzetti and Vardanega (2011) proposed a measurement-based single-thread architecture method, which calculates WCET estimates by multiplying the longest observed execution time (LOET) by the safety margin. Davis et al. (2018) designed a multicore response time analysis framework, which directly formulates the response time based on the demand for different hardware resources, so that the response time analysis is decoupled from the dependence on the context-independent WCET value. This type of WCET timing analysis mainly focuses on analyzing the influence of the hardware platform architecture or the execution process of the task itself on the running time of the task. However, this is not suitable for the timing analysis method of an autopilot system. There are various types of tasks maintained in an autopilot system, such as obstacle avoidance planning, target detection, navigation, and entertainment (Bock et al., 2018). The urgency of different tasks may vary significantly, and external environmental conditions such as weather and road sections can also affect the urgency of tasks. Among the example tasks listed previously, obstacle avoidance planning has absolute priority compared with entertainment demand. Owing to the number and variety of tasks and the requirement to protect the safety of human life, the urgency of tasks cannot be ignored in the design of autopilot systems. Unlike the task round robin scheduling based on time slice adopted by general time-sharing systems, priority is allocated to ensure that the tasks with high urgency in autonomous driving can be completed within a specified time. The autopilot system generally runs on a custom real-time operating system (Jo et al., 2014), which sacrifices some running time for less urgent tasks. When the computing resources are insufficient, the low-priority task is interrupted by the high-priority task, and the system assigns resources to the higher-priority task. Therefore, real-time systems such as autopilot systems often use preemptive scheduling algorithms based on task priority. Task preemption is the main factor that affects the execution time of tasks.

It is necessary to quantify the time fluctuation caused by task preemption when an autopilot system runs on a hardware platform, because it is essential for the design and hardware selection of an autopilot system. To solve these problems, a time analysis method is proposed for real-time applications based on a queuing theory and preemptive scheduling strategy. The main aspects of this study are as follows:

1) Based on a queuing theory and preemptive scheduling algorithm, a real-time system model suitable for automatic driving is established.

2) The time fluctuation of three subtasks of autopilot system in real software and hardware environment is obtained by testing. Analyzed the influence of the running time of each subtask in real-time application running by the preemption factor between tasks, and discussed the hardware requirements of such real-time application.

The remainder of this paper is organized as follows: Section 2 presents a discussion regarding related work, and Section 3 details the analysis of the interaction among multiple

tasks. Section 4 describes our method model, and Section 5 introduces the experimental configuration and analysis results. Finally, the paper is concluded and directions for further research are discussed in Section 6.

2 Related work

2.1 Autopilot system

Since the Eureka Project PROMETHEUS (Network, E. U. R. E. K. A., 2013), several international studies and experiments on automatic driving systems have been conducted. Among the most advanced automatic driving systems (Akai et al., 2017; Li et al., 2020; Liu et al., 2019; Maddem et al., 2017; Wei et al., 2013; Zhang & Letaief, 2019; Zhao et al., 2021), an intelligent system for a single vehicle is the most common type. A single car with such a system has automatic driving functionality even if it is not connected to the Internet. These automatic driving systems realize vehicle intelligence by modularization. Their main design concept is that different functions are divided into different modules (Behere & Torngren, 2015; McAllister et al., 2017), according to the need for driverless functionality, which finally constitute the entire automatic driving system. The core functions of modular automatic driving can be summarized as follows: synchronous positioning and mapping, perception, evaluation, planning and decision-making, vehicle control, and human-computer interface. Individual development of each module divides the difficult task of designing an autonomous driving system into a set of easy-to-solve sub-problems. At the same time, the automatic driving system has more and more functions based on modular development. For example, Bai et al. (2015) proposed an online planning method with intention awareness to estimate the unknown intentions of nearby pedestrians, to facilitate safe, effective, and stable driving. Wang et al. (2017) proposed a new moving object detection and tracking system that combines light detection and ranging measurements with visual sensor output to achieve improved performance. The WiBot (Raja et al., 2018) designed by Raja and others can detect the driver's inattentive driving behavior and realize the interaction between the driver/passenger and the vehicle by analyzing the human movements in the video data in the car, thereby improving driving safety and driving experience. The common characteristic of these functional tasks is that event response and data processing generally have strict time constraints, and the urgency of tasks is different. For example, the urgency of obstacle detection tasks is higher than that of entertainment tasks, and thus it is necessary to classify each subtask in the automatic driving system. Dai et al. (2019) categorized tasks into three levels according to their urgency, namely, extremely important applications (EIAs), very important applications (VIAs), and general important applications (GIAs) and assigned the task deadline according to the urgency of the task and the external environmental impact.

2.2 Time series analysis

At present, time series analysis methods for time-critical applications can be divided into two categories (Wilhelm et al., 2008): modeling and measurement-based approaches. Wenzel et al. (2008) and Pellizzoni et al. (2010) analyzed memory access latency in systems in which multiple parallel tasks share the main memory. These methods require a detailed analysis of the memory access patterns of the application and a deep understanding of the memory arbitration strategy. Schranzhofer et al. (2010) divided the task into a series of super blocks and used a time-division multiple access strategy to schedule tasks. The WCET of each super

block was analyzed to calculate the WCET of the task. Kelter and Marwedel (2017) proposed a technique to explore interleaving in parallel mission systems and an exclusion criterion to prove that interleaving will never occur, and they integrated this technique into existing WCET analysis methods. The aforementioned research analyzes the performance fluctuation by modeling; however, many assumptions are introduced in the modeling process, and they are difficult to realize in practical applications.

Measurement-based time series analysis is another commonly used method that mainly analyzes time changes by designing test cases and experimental cases according to variations in the hardware architecture. Mezzetti and Vardanega (2011) proposed a measurement-based single-thread architecture method that estimates the WCET by multiplying the LOET by the safety margin. Iorga et al. (2020) propose a method of measurement based on percentiles and confidence intervals, and show that it provides both competitive and reproducible observations.

However, these application timing analysis methods are not suitable for autopilot systems. In autopilot systems, the urgency of different tasks may vary significantly. For example, accident prevention has absolute priority compared with entertainment needs. Therefore, to ensure personal safety, the task of automatic driving systems adopts preemptive scheduling strategy to ensure the real-time performance of high-priority tasks. When analyzing the timing of such preemptive real-time applications, the first consideration should be the time fluctuation caused by preemption between tasks.

3 Interaction between multiple tasks

In a preemptive multitask real-time system, each subtask has its own priority. A combination of any two tasks falls into one of two states:1) A combination of tasks with different priorities, 2) A combination of tasks with the same priority. For these two different types of combinations, the impact of tasks on runtime is also different.

3.1 Tasks of different priority

In a real-time system based on a preemptive scheduling algorithm of task priority, a running low-priority task is interrupted by a newly released high-priority task when the system computing resources are insufficient, and the system does not switch back to the original task until the high-priority task is finished. In the process of task preemption, the running time of the low-priority task is mainly affected by two aspects: task preemption and context switch. Figure 1 shows an example of a running task preempted by a high-priority task and then resumed.

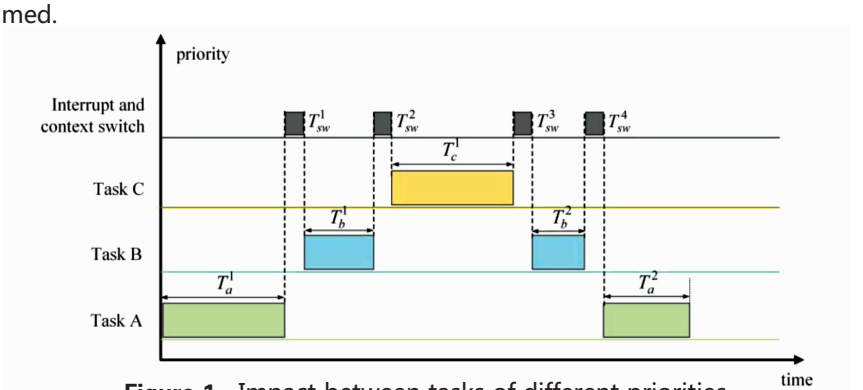


Figure 1 Impact between tasks of different priorities

The execution process of task A is described as follows: task A arrives and starts running at time 0, and task B arrives and is ready after T^1_a time. Because task B has a higher priority than task A, the system scheduler will perform a context switch, interrupts the operation of task A, and saves the corresponding state information for later recovery. The system starts to load the status information of task B and switches to task B for running. Task C arrives after T^1_b time. Because the priority of task C is higher than task B, the system generates interrupt nesting and switches to task C to run. After the time of T^1_c , task C finishes running, the scheduler restores the status information of task B, and task B resumes running from the break point. After task B has finished running, task A will resume from the break point until the task is completed. Among them, T^1_{sw} and T^2_{sw} represent the time taken to switch from task A to task B and from task B to task C, respectively, and T^3_{sw} and T^4_{sw} correspond to the recovery time of their respective tasks. Taking the lowest priority task A as an example, when no preemption occurs, the execution time of task A alone is $ET = T^1_a + T^2_a$. After preemption, the total time of task A is $ET' = T^1_a + T^2_a + T^1_b + T^2_b + T^2_a + T^1_c + \sum {}^4_{i=1} T^i_{sw}$, which is more time consumed by context switching and the running of task B and C.

It should be noted that during the execution of Task B, if a task with a higher priority arrives, interrupt preemption nesting occurs.

3.2 Tasks of same priority

If the tasks have the same priority, it can be considered that the two tasks have the same real-time requirements. Therefore, the system should be as fair as possible in the allocation of computing resources with respect to time, to avoid a situation in which a task occupies computing resources all the time. The real-time system model established in this study adopts the same scheduling method for tasks with the same priority as the time-sharing scheduling strategy. Figure 2 shows an example of interaction between two tasks of the same priority running together.

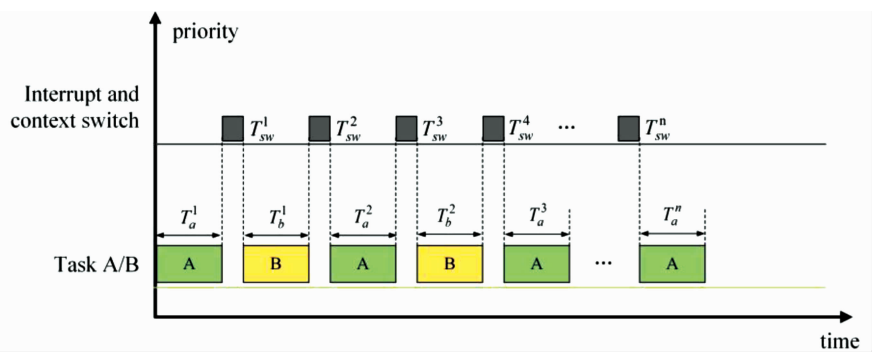


Figure 2 Impact between tasks of the same priority

For two real-time subtasks, task A and task B have the same priority, and the time slice rotation method is adopted to ensure the fair scheduling of tasks. At each time, a task is assigned to a specific time slice. When the time slice has elapsed, regardless of the priority of the thread, the task is not executed again. Instead, it enters the ready queue and waits for the arrival of the next time slice. Task B arrives after task A runs for T^1_a , the corresponding state information is saved after the time slice of task A has elapsed, and then the two tasks are executed alternately. Therefore, task A in this case has a greater proportion of the

running time of Task B and the time cost of multiple context switching compared to running alone.

4 Preemptive real–time system model

The core of the autonomous driving system can be summarized into three layers: Perception, Planning and Control. In the process of automatic driving, the perception layer and the planning layer continuously generate various tasks, and most of them are computationally intensive tasks. The unknown nature of the real road conditions leads to the randomness of the arrival of various tasks, that is, the release frequency of vehicle obstacle avoidance and target detection tasks is not fixed, but randomly distributed. Additionally, various tasks in the real-time system have different sensitivity to time, and the system prioritizes the real-time performance of more urgent tasks, so the tasks have their own preemptive priority. According to the priority preemption scheduling strategy, these published tasks are allocated to each computing core of the hardware platform. If the computing cores are full, they enter the ready queue and wait for the next assignment. We can regard these computing cores as the service desk of the service system, so the entire autopilot real-time system model can be regarded as a preemptible M/M/C queuing model with task priority. The service flow chart of the model is shown in Figure 3.

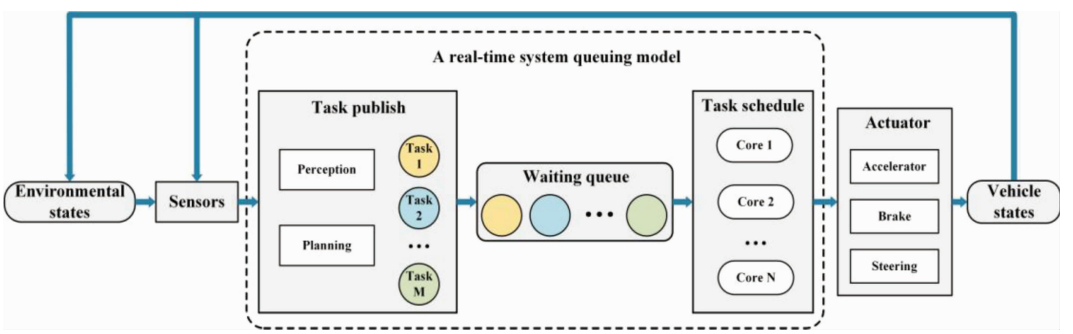


Figure 3 Preemptive real-time system based on M/M/C queuing model

The model consists of three parts: task publishing, waiting queue, and task scheduling.

4.1 Task release

4.1.1 Task selection

Automatic driving vehicles need to perceive changes in the external environment through their sensors. When processing the data collected by sensors, the vehicle computing platform produces various tasks. As shown in Figure 4, autonomous driving subtasks, including path planning, obstacle detection, navigation, and entertainment, generally have different degrees of urgency. For example, the urgency of obstacle detection is higher than that of entertainment, and thus it is necessary to classify them. According to the urgency of a task, it can be categorized into one of three levels: EIA, IA, and GA. In this study, the three types of tasks are selected and assigned different priorities as input tasks of the real-time system model.

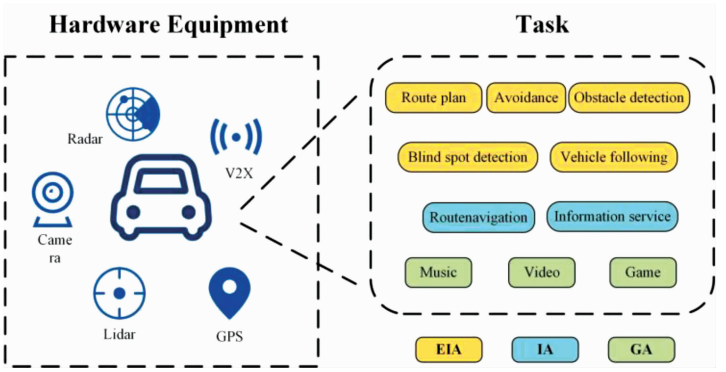


Figure 4 Classification of autonomous driving tasks

1). Local obstacle avoidance: the task is mainly used in vehicle obstacle avoidance, vehicle emergency stopping, and other scenarios. This study implements a local obstacle avoidance task based on an artificial potential field algorithm. Because the task involves personal safety, its scheduling priority is set to the highest.

2). Global path planning: this task is an important task in an autonomous driving decision-making system, mainly for vehicle route navigation services. In this study, a path planning task based on A-star algorithm is implemented. Because the vehicle should give priority to the obstacle avoidance task, the priority of the path planning task is set as medium.

3). Entertainment task: this type of task is an application program related to the passenger riding experience. This study simulates the operation of this type of task through the program, and the priority of this task is set as the lowest.

4.1.2 Task arrival rules

The arrivals of the three types of tasks are independent and identically distributed. The arrival process of each type of task is a Poisson flow input process, and the arrival intervals of the tasks follow a negative exponential distribution with λ_1 , λ_2 , λ_3 , as follows:

$$P\{T_i < x\} = 1 - e^{\lambda i x}, x > 0 \quad (1)$$

Here, λ_i represent the average numbers of arrivals of the i^{th} task in a unit time. From this, three independent and identically distributed random variable sequences can be generated: $\{U^1_k, k \geq 1\}$, $\{U^2_k, k \geq 1\}$, $\{U^3_k, k \geq 1\}$,U^1_k (i=1,2,3) ,represents the arrival interval between the $(k-1)^{th}$ task and the k^{th} task of the i^{th} task. Let S^i_k be the arrival time of the k^{th} task of the i^{th} task; then:

$$S^i_1 = U^i_1, S^i_k = U^i_1 + U^i_2 + \ldots + U^i_k \quad (2)$$

We can determine the specific arrival time of each of the three types of tasks through Formula (2).

4.2 Wait queue

Most of the tasks generated during the operation of the autonomous driving system will not be executed immediately. When a task arrives in the system, if all the service desks are busy and the priority of the task is not sufficiently high for preemption, the task enters the waiting queue and wait for the next system scheduling. Tasks with Three types of tasks are stored in the waiting queue, with respect to their priority, which are divided into the following two situations according to the service progress:

1). Since its release, it has never been scheduled to run by the system, and the service progress of such tasks is 0.

2). When service is interrupted because it is preempted by a higher-priority task, it re-enters the queue to wait for the next scheduling. The service progress of such tasks is saved; that is, the service does not need to be restarted for the next run.

The real-time system model established in this paper adopts the waiting system queuing rules, and the queue length is unlimited. The task will remain in the ready queue until it is scheduled and executed by the scheduler.

4.3 Task scheduling and running

At present, the main computing platform architecture adopted by the autonomous driving system is basically Intel's X86, which generally has multiple execution units and can execute tasks in parallel. The task scheduler is mainly responsible for scheduling the issued priority tasks in the ready queue to the corresponding hardware execution unit to run. The scheduler of the real-time system adopts a preemptible scheduling strategy with task priority. When a task with a higher priority arrives in the system, regardless of the service progress of the task being run, the service must be suspended and replaced by this high-priority task service. Because tasks of the same priority may appear, the service rules of the model are divided into the following two types according to the priority combination:

1). Different priority tasks: High-priority tasks have interrupt preemption rights over low-priority tasks. For example, an IA-level task in the system is running, and an EIA-level task is released at this time. If there is an idle service desk, the system immediately serves the EIA-level task; if there is no idle service station, the running IA-level task will be interrupted and returns to the waiting queue. The service desk starts to serve the EIA-level tasks. When the EIA-level tasks are completed, the system traverses the waiting queue, calculates the scheduling weight according to the priority, and selects the task with the highest weight to run.

Algorithm 1 : Preemptive scheduling algorithm

```
Input：waiting queue , Q ; newly arrived task , task_new ; running task,task_now
Output：the next task to be executed,task_next
1 : if Priority(task_new) > Priority(task_now) then
2 :    intertupt(task_now);
3 :    addToQueueHead(task_now,Q);
4 :    task_next=task_new;
5 : else
6 :    addToQueueHail(task_new,Q);
7 :    foreach task ∈ Q do
8 :       calculateSchedulingWeight(task)
9 :       if remainTime(task)==0 then
10 :        addToQueueTail(task);
11 :        resetRemainTime(task);
12 :     end if
13 :    end for
14 :    task_next=findMaxWeightTask(Q);
15 : end if
16 : return task_new;
```

2). Tasks of the same priority: Fair scheduling is maintained between tasks of the same priority. A time-sharing scheduling strategy is adopted between tasks. When a task's current time slice has elapsed, the task actively gives up computing resources and is placed at the end of the waiting queue until the next cycle is re-divided into time slices before continuing. Then, the system selects task at the head of the queue with the same priority to start running.

The general flow of the preemptive scheduling algorithm corresponding to the service rule is shown in Algorithm 1.

5 Experiments and results

5.1 Experimental environment and configuration

The hardware processor mainly used in the experiment is the Intel Core i5-8400, which has six physical cores, does not support hyper-threading technology, and executes up to six thread tasks simultaneously. We assign the following functions to its cores:

1). Core1: the publisher that runs local obstacle avoidance tasks, and the task release interval obeys the negative exponential distribution of parameter λ_1. The actual operation of the local obstacle avoidance task is the classical obstacle avoidance algorithm in automatic driving: the artificial potential field method. The program realizes a fixed start point, end point, and a local path planning operation under the map.

2). Core2: the publisher that runs the path planning task, and the task release interval obeys the negative exponential distribution of the parameter λ_2. The path planning task uses a heuristic search algorithm: A-star path finding algorithm. A single task completes the optimal path search operation with a fixed starting point and ending point under the same cost map.

3). Core3: A publisher that runs entertainment tasks, and the task release interval obeys the negative exponential distribution of the parameter λ_3. The entertainment task implemented in this experiment is a simulation task, and its running time is longer than the local obstacle avoidance task and the global path planning task.

This experiment runs on the mature Linux operating system (Ubuntu 16.04). The release of a task is equivalent to the creation of the corresponding task thread, and the completion of the task is equivalent to the destruction of the task thread. At the same time, the processes of publishing tasks by the three task publishers are independent of each other. To avoid the migration of task publishing threads between different cores of the processor, we use the sched setaffinity system call to bind the publishers of the three types of tasks to the corresponding cores. The remaining cores, Core4, Core5, and Core6, are used as optional task operation cores.

Due to the large difference in the running time of the three tasks, in order to eliminate the influence of measurement scale and dimension, this paper uses the "coefficient of variation (CV)" to measure the degree of dispersion of each group of data. The calculation formula is as follows:

$$CV = \frac{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2}}{n\bar{X}} \times 100\% \quad (3)$$

where $\bar{X}$ is the mean of sample X.

5.2 Comparison of preemptive real–time system and general time–sharing system

First, the running times of the three types of tasks described in Section 4.1 are measured, and the measurement result is the average running time of the task executed 100 times, as shown in Table 1.

Table 1 Single task running time.

Task Name	Time
Local obstacle avoidance	35.1ms
Global path planning	0.24s
Entertainment	1.26s

For the time-sharing system, the Linux default process scheduling strategy SCHED OTHER is adopted. To quantify and compare the difference in volatility of task sets on the two types of scheduling systems, we fixed the probability distribution parameters of the above three types of tasks as:λ_1=1.4, λ_2=0.9,λ_3=0.4; and at the same time, the number of schedulable cores is set to 1. Only adjusting the scheduling strategy of the system, we record the time changes of three different priority tasks for 200 cycles in this experiment.

As shown in Figure 5(a), on a preemptive real-time system, the average running time of the local obstacle avoidance task with the highest priority is approximately the same as when it runs alone. On the general time-sharing system, the average running time increased by a factor of 5.24. Compared with general time-sharing systems, the average time consumed by preemptive real-time systems has been reduced by 80.8%. The coefficients of variation are CV_{RS}=13.5% and CV_{TS}=78.0% . The task stability of the preemptive real-time system is significantly better than that of the time-sharing system.

Figure 5(b) shows a comparison of the running time of the global path planning task on the two types of systems. When running on a preemptive real-time system, the average running time is 1.29 times that of a single running system. On the general time-sharing system, its average running time increased by a factor of 5.67. The average time consumption of preempting real-time systems is 77.2% lower than that of general time-sharing systems. The coefficients of variation are CV_{RS}=28.1% and CV_{TS}=71.7, and the stability gap between the two types of systems is reduced.

For the entertainment task with the lowest priority, as shown in Figure 5(c), the task runs longer on the preemptive system than on the general time-sharing system, and the average time consumption increases by 20.7%. The running time on the preemptive system is 4.71 times that of the stand-alone operation, and that of the general time-sharing system is 4.12 times that of the stand-alone operation. Considering that the entertainment task has no urgent requirements for running time, it is still within the acceptable range of the system. The coefficients of variation are CV_{RS}=69.2% and CV_{RS}=77.1% , and the volatility is similar.

Overall, for tasks with higher priority, preemptive real-time systems have more advantages over general time-sharing systems in terms of real-time and stability requirements. However, for tasks with lower priority, the running time may be more time-consuming than on the time-sharing system. Therefore, in the priority allocation process of a subset of real-time application tasks, the urgency of each task must be fully considered.

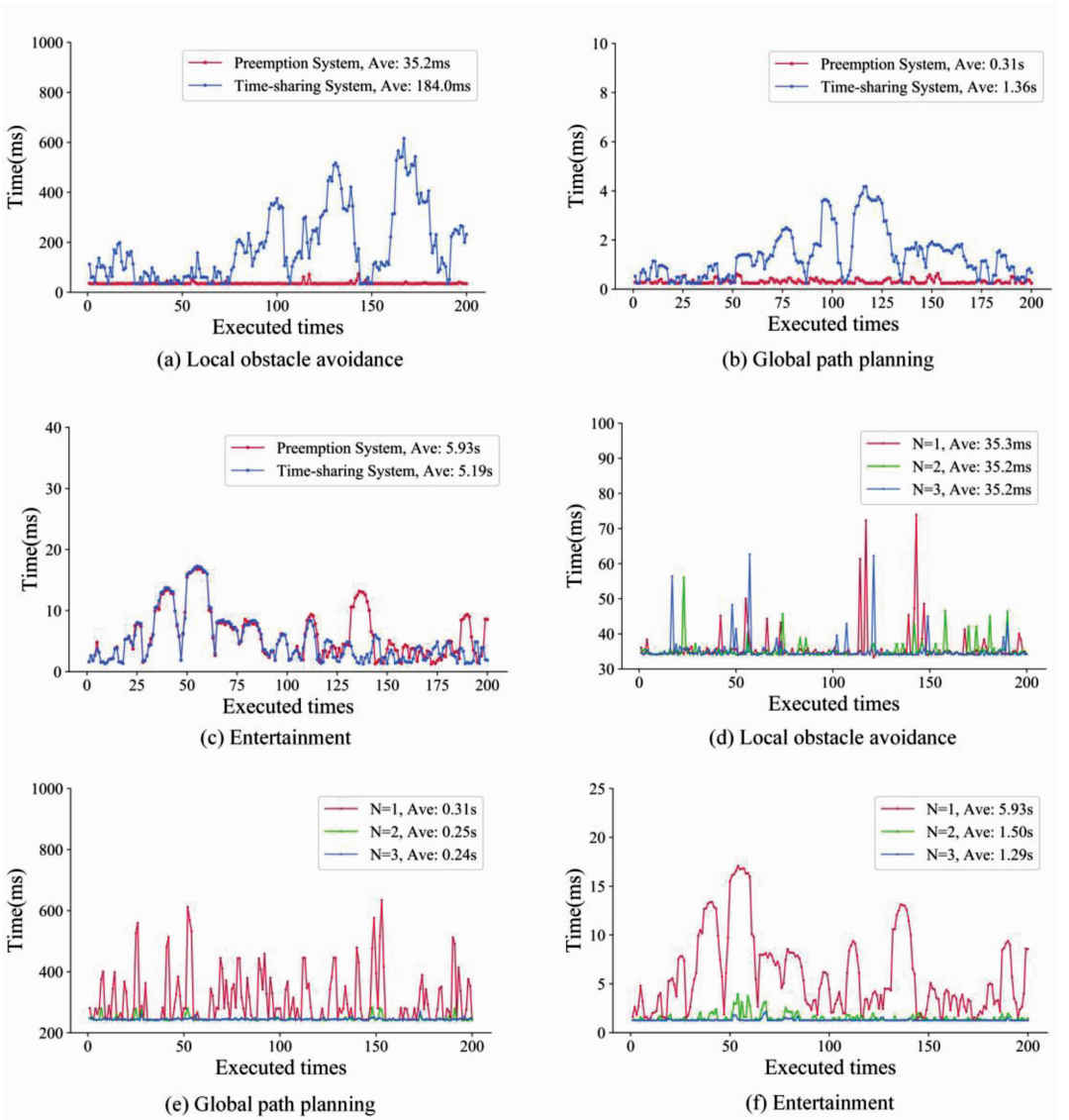


Figure 5 Time analysis under different conditions.

5.3 Analysis of the impact of schedulable cores on real–time applications

The number of schedulable cores of the hardware platform for actual real-time application deployment is generally more than one, that is, subtasks can run in parallel. The experiment recorded the running time of each priority task when the number of schedulable cores N=1, 2, 3 under the preemptive real-time system by adjusting the number of schedulable cores. The parameters of the arrival distribution of the three types of tasks are fixed as: λ_1=0.8, λ_2=0.3,λ_3=0.08, and the scheduling method adopts preemptive scheduling with priority. The results are shown in Figure 5(d, e, f).

Calculating the ratio of the average time to the time when running tasks alone at all levels under different number of scheduling cores:

Table 2 Impact of the number of schedulable cores on task time

Number of tasks and schedulable cores	Task Names		
	Local obstacle avoidance	Global path planning	Entertainment
N=1	1.01	1.29	4.71
N=2	1	1.04	1.19
N=3	1	1	1.02

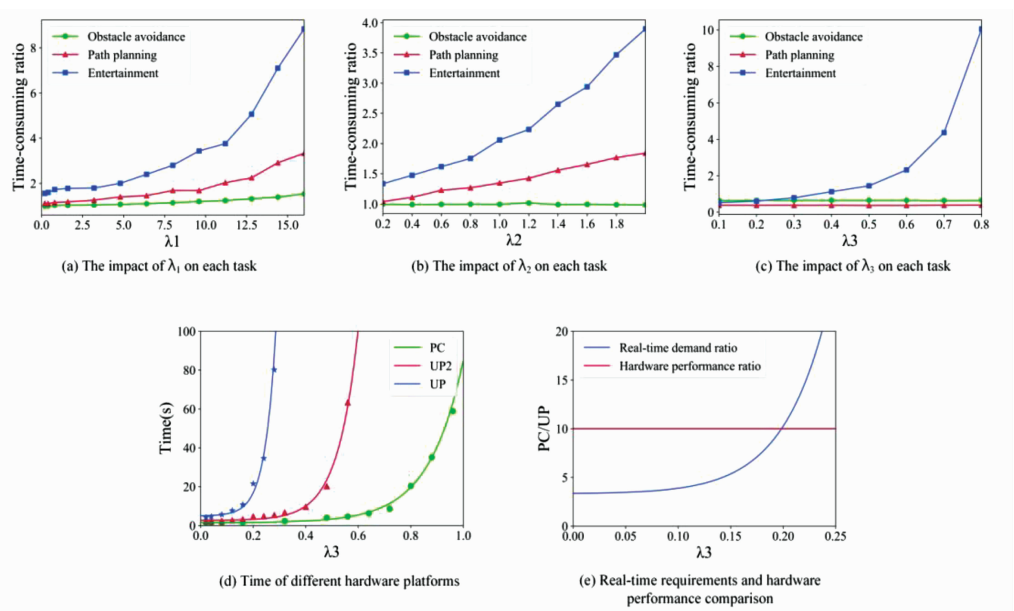
As shown in Table 2, as the number of schedulable cores N increases, the real-time performance of tasks of each priority is improved. For tasks of different priorities, the lifting effect is different. For the local obstacle avoidance task with the highest priority, because its real-time performance under the real-time system is good, the improvement effect is not obvious. For the entertainment task with the lowest priority, the average time consumption when the number of scheduling cores is 3 is 78.3% lower than that when the number of scheduling cores is 1, and the real-time performance is improved the most.

5.4 Impact of task arrival distribution on real–time systems

When an autonomous vehicle is driving under different road conditions, the frequency of each subtask may be different. For example, when driving on a typical urban road, the number of obstacle avoidance tasks increases significantly. In this section, by adjusting the distribution parameter λ_i of each subtask publisher, we study the impact of the arrival distribution of each priority task on real-time applications. The experiment is carried out considering following aspects:

5.4.1 Impact of λ_1 on the running time of tasks at all levels

We the fixed global path planning and entertainment task arrival distribution parameters: λ_2=0.4, λ_3=0.2. To make the experiment results more evident, we set the number of schedulable cores N = 1, and the experiment records the ratio of the average running time of tasks at different λ_1 to the individual running time.



(a) The impact of λ_1 on each task (b) The impact of λ_2 on each task (c) The impact of λ_3 on each task

(d) Time of different hardware platforms (e) Real-time requirements and hardware performance comparison

Figure 6 Analysis of hardware requirements for real-time systems

As shown in Figure 6(a), with the increase in λ_1, the number of local obstacle avoidance tasks per unit time increases, and the running time of the three priority tasks increases, but the impact intensity of tasks at all levels is different. The local obstacle avoidance task is the highest-priority task, and its running time is mainly affected by the round-robin scheduling between tasks at the same level, such that its performance is less degraded. The priority of global path planning tasks is lower than that of local obstacle avoidance tasks, and the probability of being interrupted by obstacle avoidance tasks increases during operation, such the performance degradation is more obvious than that for obstacle avoidance tasks. For the entertainment task with the lowest priority, because obstacle avoidance and path planning tasks interrupt its operation, the average running time increases the most.

5.4.2 Impact of λ_2 on the running time of tasks at all levels

The arrival distribution parameters of the obstacle avoidance and path planning tasks are λ_1=0.2, λ_3=0.2, and the number of scheduling cores is 1. The ratio of the average running time of all levels of tasks under different λ_2 to individual running was recorded.

It can be seen from Figure 6(b) that as λ_2 increases, the average running time of path planning tasks and entertainment tasks increases. Because the path planning task interrupts the entertainment task, the performance of the entertainment task declines even more. The local obstacle avoidance task is the highest-priority task, and the system gives priority to its operation. Therefore, the increase in λ_2 does not affect the real-time performance of the local obstacle avoidance task.

5.4.3 Impact of λ_3 on the running time of tasks at all levels

The distribution parameter of mission arrival probability of the obstacle avoidance and path planning tasks are λ_1=0.2, λ_2=0.4, and the number of schedulable cores is N=1. The experiment records the ratio of the average running time of all levels of tasks with different sizes of λ_3 to the individual running. The result is shown in Figure 6(c).

The increase in λ_3 means that the number of entertainment tasks reached per unit time increases. Because the priority of the obstacle avoidance and path planning tasks is higher than that of the entertainment task, the running time of both is not affected by the change in λ_3 .

Generally, in the preemptive real-time system with task priority, the change in the task arrival probability of a single subtask affects all subtasks whose priority is less than or equal to that task, and the lower the subtask priority is, the greater the performance degradation is. On the contrary, the subtasks with higher priority are almost unaffected.

5.5 Impact of hardware platform performance on real–time applications

From the experiment presented in Section 5.4, we know that when the task probability distribution parameter λ increases, the average time between arrivals of two sequential tasks decreases, and the average running time of the corresponding tasks increases. When λ increases to a certain value, a task may not be completed on time owing to excessive releases. In this case, it is necessary to consider replacing hardware devices with higher-performance devices to ensure the reliability of the system.

To study the impact of hardware platform performance on the task running time of preemptive real-time systems, we considered the lowest priority, that is, the entertainment task with the most apparent time change as the task under study, and conduct comparative experiments on three hardware platforms with different performance, PC, UP Board(UP), UP Squared(UP2). First, the independent running time of each subtask on the three platforms is

used as the performance reference of the platform, and the results are shown in Table 3.

Table 3 Individual running time of each task under different hardware platforms

Platforms	Task Names		
	Local obstacle avoidance	Global path planning	Entertainment
PC	35.1ms	0.24s	1.26s
UP Squared	94.2ms	0.50s	1.91s
UP Board	240.6ms	1.02s	3.20s

Taking into account the respective priorities of the tasks, we set the performance weights of the three types of tasks to local obstacle avoidance: global path planning: entertainment = 3: 2: 1, and the comprehensive performance ratio of the three platforms is calculated as PC: UP2: UP = 5.2: 2.3: 1.

Second, to analyze the performance differences of the preemptive real-time system on the three hardware platforms, the following experimental configurations are adopted: fixed local obstacle avoidance tasks and path planning tasks. The task arrival probability distribution parameters are λ_1=0.3 and λ_2=0.1, and the the number of schedulable cores is N = 1, The average running times of the first 200 cycles of entertainment tasks under different λ_3 on the three platforms are recorded.

From Figure 6(d), we found that under the same task probability distribution, the running time of entertainment tasks on different hardware platforms are different, but they all approximately conform to the law of exponential change. We try to fit the three curves with the exponential regression equation model: $Y=ae^{bx}+c$ and the fitting method adopts the robust least-squares fitting to minimize the absolute residual. Then, the fitting results are: $Y_{PC}=0.042e^{7.62x}+1.46$,$Y_{UP2}=0.042e^{12.95x}+2.62$, and $Y_{UP}=0.095e^{24.19x}+4.96$. The R-square values of the three fitting curves are calculated as Rs_{PC}=0.9994, Rs_{UP2}=0.9946, Rs_{UP}=0.9913; as these are all close to 1, the fitting effect is good.

Using the above fitting curve, we can evaluate the applicable scope of different hardware platforms. When the running time of our task is required to be fixed, calculate the range of λ_3 through the inverse function $x=\frac{\ln(y-c)-\ln a}{b}$ of $Y= ae^{bx}+ c$. For example, when the entertainment task requires the average running time not to exceed 20s, the maximum λ_3 corresponding to the three platforms is calculated: λ_{3PC}=0.70, λ_{3UP2}=0.40, λ_{3UP}=0.16 . In scenarios where entertainment tasks are sparse, such as once every 10s on average, the three platforms can meet the real-time requirements of entertainment tasks. If it happens every 2s on average, then only the PC can meet its real-time requirements.

Figure 6 (e) analyzes the relationship between real-time requirements for entertainment tasks and hardware performance requirements. With the increase of task intensity, the number of tasks queued for processing gradually increases, and the improvement of hardware performance alone will become increasingly unable to meet the real-time requirements of system tasks.

6 Conclusion

Currently, the widely used time sequence analysis methods cannot analyze the time changes caused by preemption among subtasks in preemptive real-time applications such as

autonomous driving, and thus cannot effectively evaluate the real-time performance of edge platforms running real-time applications. In this study, by setting the priority of each subtask in a real-time system, the running process of the application is transformed into three parts, which are the task arrival, queue waiting, and scheduling service in the queuing model, and a real-time application based on a queuing theory and preemptive scheduling strategy is established. Three commonly used subtasks in an automatic driving system are used as test tasks, and the time fluctuations of the three subtasks in the real software and hardware environment are tested; the following conclusions are obtained.

1) The real-time system with task preemption priority can better meet the real-time requirements of higher-priority tasks. However, for tasks with lower priority, its performance may not be as good as that of the general time-sharing system.

2) By increasing the number of schedulable cores in the real-time system, the real-time performance of tasks at all levels can be effectively improved, and the lower the priority, the more evident is the improvement effect.

3) The change in the arrival probability distribution of a single subtask affects the running time of all subtasks with a lower priority, and the lower the task priority, the greater the time change.

4) The improvement in hardware platform performance can improve the real-time performance of tasks to a certain extent. However, when the intensity of tasks increases, the improvement in hardware performance cannot meet the real-time requirements of the system.

Acknowledgements

The work is supported by the National Key Research and Development Program under Grant No. 2019YFC0118404, the National Natural Science Foundation of China under Grant No. U20A20386, the Zhejiang Key Research and Development Program under Grant No. 2020C01050, the Key Laboratory fund general project under Grant No. 6142110190406, the Zhejiang Natural Science Foundation Project under Grant No.LY19F020044.

References

Akai, N., Morales, L. Y., Yamaguchi, T., Takeuchi, E., Yoshihara, Y., Okuda, H., ... & Ninomiya, Y.(2017). Autonomous driving based on accurate localization using multilayer LiDAR and dead reckoning. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems(ITSC)*(pp.1–6). IEEE.

Bai, H., Cai, S., Ye, N., Hsu, D., & Lee, W. S.(2015). Intention–aware online POMDP planning for autonomous driving in a crowd. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 454–460). IEEE.

Behere, S., & Torngren, M.(2015). A functional architecture for autonomous driving. In *2015 First International Workshop on Automotive Software Architecture (WASA)* (pp. 3–10). IEEE.

Bock, F., Siegl, S., Bazan, P., Buchholz, P., & German, R.(2018). Reliability and test effort analysis of multi–sensor driver assistance systems. *Journal of Systems Architecture, 85,* 1–13.

Dai, H., Zeng, X., Yu, Z., & Wang, T.(2019). A scheduling algorithm for autonomous driving tasks on mobile edge computing servers. *Journal of Systems Architecture, 94*, 14–23.

Davis, R. I., Altmeyer, S., Indrusiak, L. S., Maiza, C., Nelis, V., & Reineke, J.(2018). An extensible framework for multicore response time analysis. *Real–Time Systems, 54*(3), 607–661.

Iorga, D., Sorensen, T., Wickerson, J., & Donaldson, A. F. (2020). Slow and steady: Measuring and tuning multicore interference. In *2020 IEEE Real–Time and Embedded Technology and Applications Symposium (RTAS)* (pp. 200–212). IEEE.

Jo, K., Kim, J., Kim, D., Jang, C., & Sunwoo, M. (2014). Development of autonomous car–Part I: Distributed system architecture and development process. *IEEE Transactions on Industrial Electronics, 61* (12), 7131–7140.

Kelter, T., & Marwedel, P. (2017). Parallelism analysis: Precise WCET values for complex multi–core systems. *Science of Computer Programming, 133,* 175–193.

Li, L., Zheng, N., & Wang, F. Y. (2020). A theoretical foundation of intelligence testing and its application for Intelligent Vehicles. *IEEE Transactions on Intelligent Transportation Systems.*

Li, P., Zhao, H., Liu, P., & Cao, F. (2020). RTM3D: Real–time monocular 3D detection from object keypoints for autonomous driving. arXiv preprint arXiv:2001.03343, 2.

Lim, W., Lee, S., Sunwoo, M., & Jo, K. (2019). Hybrid trajectory planning for autonomous driving in on–road dynamic scenarios. *IEEE Transactions on Intelligent Transportation Systems.*

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

Maddern, W., Pascoe, G., Linegar, C., & Newman, P. (2017). 1 year, 1000 km: The Oxford RobotCar dataset. *The International Journal of Robotics Research, 36* (1), 3–15.

McAllister, R., Gal, Y., Kendall, A., Van Der Wilk, M., Shah, A., Cipolla, R., & Weller, A. (2017). Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning. *International Joint Conferences on Artificial Intelligence,* Inc.

Mezzetti, E., & Vardanega, T. (2011). On the industrial fitness of wcet analysis. *In Proceedings of the 11th International Workshop on Worst–Case Execution–Time Analysis (WCET2011)*, Ed. Austrian Computer Society (OCG)

Network, E. U. R. E. K. A. (2013). Programme for a European traffic system with highest efficiency and unprecedented safety.

Pellizzoni, R., Schranzhofer, A., Chen, J. J., Caccamo, M., & Thiele, L. (2010). Worst case delay analysis for memory interference in multicore systems. *In 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)* (pp. 741–746). IEEE.

Raja, M., Ghaderi, V., & Sigg, S. (2018). WiBot! In–vehicle behaviour and gesture recognition using wireless network edge. *In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (pp. 376–387). IEEE.

Schranzhofer, A., Chen, J. J., & Thiele, L. (2010). Timing analysis for TDMA arbitration in resource sharing systems. *In 2010 16th IEEE Real–Time and Embedded Technology and Applications Symposium* (pp. 215–224). IEEE.

Wang, C., Zhu, Y., Jiang, J., Qiu, M., & Wang, X. (2017). Dynamic application allocation with resource balancing on NoC based many–core embedded systems. *Journal of Systems Architecture, 79,* 59–72.

Wei, J., Snider, J. M., Kim, J., Dolan, J. M., Rajkumar, R., & Litkouhi, B. (2013). Towards a viable autonomous driving research platform. *In 2013 IEEE Intelligent Vehicles Symposium (IV)* (pp. 763–770). IEEE.

Wenzel, I., Kirner, R., Rieder, B., & Puschner, P. (2008). Measurement–based timing analysis. *In International Symposium on Leveraging Applications of Formal Methods, Verification and Validation* (pp. 430–444). Springer, Berlin, Heidelberg.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., ... & Stenström, P. (2008). The worst–case execution–time problem–overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3), 1–53.

Wilhelm, R., Pister, M., Gebhard, G., & Kästner, D. (2020). Testing implementation soundness of a WCET analysis tool. In *A Journey of Embedded and Cyber–Physical Systems* (pp.5–17). Springer, Cham.

Yurtsever, E., Lambert, J., Carballo, A., & Takeda, K. (2020). A survey of autonomous driving: Common prac–

tices and emerging technologies. *IEEE Access, 8,* 58443–58469.

Zhang, J., & Letaief, K. B.(2019). Mobile edge intelligence and computing for the internet of vehicles. *Proceed–ings of the IEEE, 108* (2), 246–261.

Zhao, C., Li, L., Pei, X., Li, Z., Wang, F. Y., & Wu, X.(2021). A comparative study of state–of–the–art driving strategies for autonomous vehicles. *Accident Analysis & Prevention, 150,* 105937.