HCPerf: Driving Performance-Directed Hierarchical Coordination for Autonomous Vehicles

Jialiang Ma[†1], Li Li[†1], Zejiang Wang[2], Jun Wang[3], and Chengzhong Xu[1]

[1]University of Macau, IOTSC
[2]Oak Ridge National Laboratory
[3]Futurewei Technologies

***Abstract*—The rapid development of autonomous driving poses new research challenges to the on-vehicle computing system. In particular, the execution time of autonomous driving tasks highly depends on the specific driving environment. For instance, the execution time of configurable sensor fusion increases significantly as the scene becomes complex, which leads to end-to-end deadline misses from sensing to control and may cause accidents. Thus, a framework that can effectively utilize the system resources to guarantee the end-to-end deadlines of autonomous driving tasks as well as effectively prioritize the responsiveness and throughput of the control commands is crucial for autonomous driving.**

In this paper, we propose HCPerf, a performance-directed hierarchical coordination framework that intelligently coordinates the autonomous driving tasks with high execution time variation and complex dependencies according to the driving performance in real-time. Specifically, HCPerf mainly consists of two coordinators. The internal coordinator intelligently schedules the tasks according to the driving performance of the vehicle in order to help them meet the end-to-end deadlines while well prioritizing the responsiveness and throughput of the control commands. At the same time, the external coordinator dynamically tunes the rates of tasks according to the schedulability in order to efficiently utilize the system resource. We conduct extensive experiments on both simulation and hardware testbeds with the representative autonomous driving application. The results show that HCPerf can effectively improve the driving performance by 7.69%-45.94% in different driving scenarios.

***Index Terms*—Autonomous Driving, Real-Time Scheduling.**

I. Introduction

The advent of autonomous driving (AD) is an important milestone in the automotive industry. In the meantime, the development of various autonomous driving applications (e.g., automated vehicle path planning, automated parking and obstacle avoidance) makes the autonomous driving system become increasingly complex. For instance, Apollo Cyber RT [1], a runtime framework designed specifically for autonomous driving, contains a large number of tasks with complex dependencies. Moreover, it is expected that the number of embedded tasks will keep increasing [2]. However, given that the cost of the computing unit already accounts for 33% of the whole autonomous driving vehicle [3], the cost-conscious vehicle manufacturers are unlikely to keep adding extra computing resources to the vehicle. The increasing number of tasks with complex dependencies coupled with limited computing resources are posing new research challenges to the management of autonomous driving systems.

Limitation of Existing Approaches. Responsiveness and throughput of the control commands are two critical factors that directly impact the driving behavior of autonomous vehicles [4]. Responsiveness dictates how quickly the control commands can be generated in particular driving scenarios (e.g., emergency cases) in order to avoid potential accidents. On the other hand, the throughput quantifies the number of control commands sent to the actuator per second. A higher throughput allows for smoother control without abrupt turns and brakes, providing a better passenger experience [5]. In autonomous vehicles, valid control commands can be generated only when the end-to-end deadlines of the tasks can be met. Thus, guaranteeing the tasks meet their end-to-end deadlines and prioritizing the responsiveness and throughput of the control commands is critical for autonomous driving. Traditional embedded systems use static scheduling methodologies [6]–[9] to guarantee real-time performance. These methodologies estimate the worst-case execution times of tasks offline and then schedule tasks for execution in open-loop manner. However, the execution times of autonomous driving tasks have large variances which highly depend on the runtime inputs [2]. For instance, the execution time of configurable sensor fusion is decided by the amount of detected objects and can vary significantly in different driving environments. This means that the execution time is hard to measure precisely offline. Though the worst-case execution time of the tasks can be overestimated and the scheduling can be conducted in a conservative manner, more computing resources need to be added to the vehicle which leads to extra cost [10]. Thus, the static or open-loop scheduling methods are not efficient for autonomous driving.

On the other side, adaptive real-time scheduling is proposed to dynamically adjust the task rate in a closed-loop manner through monitoring the schedulability online in order to react to the runtime execution time variation [11]–[13]. Though these approaches can guarantee the end-to-end deadline in traditional real-time distributed systems, they are still not efficient for autonomous driving. This is for the reason that monitoring schedulability (e.g., system utilization) alone cannot effectively tell the *right* time to generate control commands in order to prioritize the responsiveness and throughput. For instance,

[†]Equal Contribution.

for the car following application [14], when an emergency braking of the car in the front is detected, a large amount of control commands are required in a short period in order to generate responsive reactions [1]. Consequently, a task-scheduling framework that can effectively guarantee the end-to-end deadlines of the tasks while intelligently prioritizing the responsiveness and throughput of the control commands without requiring extra computing resources is crucial for autonomous driving.

Observation and Challenge. Designing such a framework is not straightforward as autonomous driving systems face a totally different set of challenges than traditional real-time systems. First, traditional scheduling schemes based on system utilization [15] or acceptance ratio [6] are not efficient for autonomous driving as they cannot effectively differentiate the driving scenario (e.g., emergency situation) in which a large amount of control commands are urgently required. Thus, the driving scenario should be a critical metric to take into account. In addition, a chain of tasks including sensing, perception, prediction, planning and control should be sequentially and periodically completed before the corresponding deadlines in order to successfully generate control commands in autonomous driving. Different tasks released in various control periods can coexist in the ready queue. Though allocating computing resources to the control-related tasks (e.g., speed control) can immediately generate valid control commands, at the same time it may cause the tasks that have a tighter deadline (e.g., sensing or perception) to miss the deadline which reduces the overall throughput. Thus, how prioritizing the responsiveness and throughput of the control commands according to the driving scenario is a critical challenge for autonomous driving.

In this paper, we propose HCPerf, a driving performance-directed hierarchical coordination framework for autonomous vehicles. HCPerf uses the configured high-level performance metric to differentiate various driving scenarios and helps the operating system effectively utilize system resources to guarantee the tasks meet their end-to-end deadlines while prioritizing the responsiveness and throughput of the control commands accordingly. HCPerf consists of two coordinators, the internal coordinator and the external coordinator. Specifically, the internal coordinator keeps monitoring the configured driving performance metric of the vehicle (e.g., tracking error for car following) and adaptively schedules the autonomous driving tasks. For instance, when the tracking error increases, HCPerf schedules the tasks to generate sufficient amounts of control commands as soon as possible in order to generate a responsive reaction to mitigate the tracking error and avoid possible accidents. On the other hand, when the tracking error is small, HCPerf conducts scheduling to improve the overall throughput for achieving smoother control of the vehicle. Meanwhile, the external coordinator monitors the schedulability of the system and reduces end-to-end deadline misses while efficiently utilizing system resources. To our best knowledge, HCPerf is the *first* work that uses the end-to-end runtime driving performance to direct the task scheduling and resource management in order to balance the real-time, responsiveness and throughput of the system. Specifically, this paper makes the following three major contributions:

- We identify a new research challenge in real-time scheduling of the autonomous driving system that is introduced by the specific characteristics of autonomous driving control.
- We design HCPerf, a hierarchical coordination framework that overcomes the limitations of existing solutions by intelligently considering the driving performance, the dependencies of different tasks, and the system load.
- We evaluate HCPerf with the representative autonomous driving application on both simulation and hardware testbeds. Our results demonstrate that HCPerf effectively improves the driving performance by 7.69%-45.94% compared with the state-of-the-art and state-of-the-practice approaches.

II. MOTIVATION

In this section, we use car following as a case study to motivate the design of HCPerf.

Car following [14] is a representative application in autonomous driving that controls an autonomous vehicle's behavior according to the lead vehicle in the same lane. The workflow can be mainly divided into the following steps: 1) the autonomous vehicle detects the speed of the lead vehicle through sensors such as camera and lidar; 2) it calculates the speed to be reached through the data fusion and prediction modules; 3) the planning module transmits the planned trajectory to the control module; 4) the control module generates and sends a control command to the chassis for the corresponding operation.

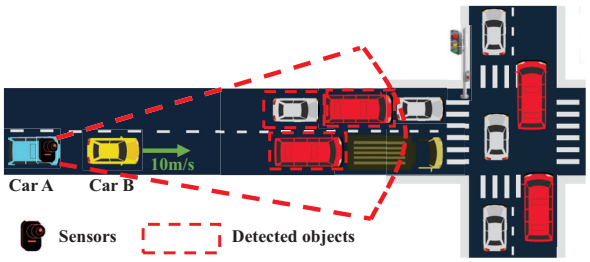


Fig. 1. A possible scenario to cause execution time increase in the autonomous driving system.

Figure 1 shows the specific scenario of the motivation example. In particular, car A is an autonomous vehicle in autonomous driving mode with the car following feature enabled to follow a human-driven car B on an urban road. At first, car A and car B run at a constant speed of 10m/s on a city road. After 5 seconds of driving, the driver of car B observes a red light 200m ahead and starts to slow down the vehicle. At the same time, the trajectory of car B and other obstacles (e.g., a large number of vehicles waiting for traffic lights and pedestrians at the intersection) are detected by car A through the camera and Lidar.

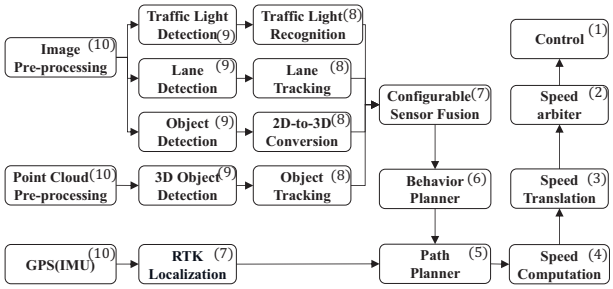


Fig. 2. Examples of autonomous driving tasks and their dependencies. The number in brackets represents the corresponding priority.

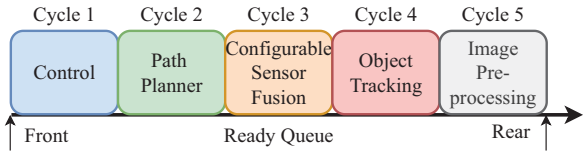
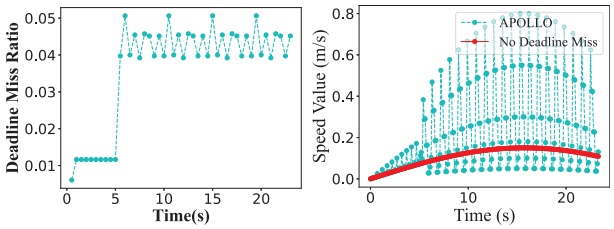


Fig. 3. An example of the task queue. Tasks from different control cycles (indicated by the top of each block) can coexist in the ready queue.

In order to examine this scenario, we simulate the basic functions of an autonomous vehicle including image pre-processing, traffic light detection and configurable sensor fusion. Figure 2 shows the simulated tasks and their dependency. The tasks are periodically released and pushed into the ready queue. Thus, the ready queue may contain tasks that belong to different release cycles. Figure 3 shows an example of the ready queue. The scheduling policy in Apollo Cyber RT [1] is adopted to schedule those tasks, in which each task is statically assigned a priority value. In particular, the priorities are designed according to the dependency among the tasks. The lower the value is, the higher the priority of the task is. In this case, Control has the highest priority. The task that has the highest priority in the ready queue will be first executed.

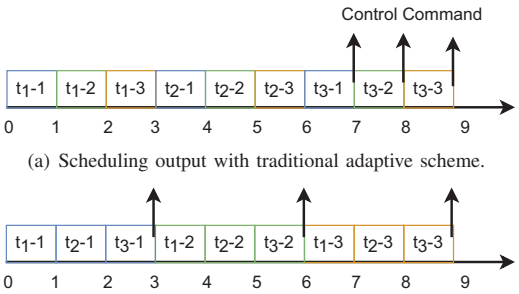
In the simulated scenario, we found that the number of detected obstacles significantly affects the processing time of the configurable sensor fusion task. This is because it uses the Hungarian algorithm [16], with time complexity $O(n^3)$ for data matching. Thus, its execution time is highly dependent on the number of obstacles (n) detected at runtime.



(a) Deadline Miss Ratio. (b) Speed Tracking Error.

Fig. 4. Motivation experiment results.

Given the execution time variation of configurable sensor fusion, the corresponding deadline miss ratio for the autonomous vehicle (Car A) is represented in Figure 4(a). We can see that the deadline miss ratio starts to increase after 5 seconds caused exactly by the increased execution time of the configurable sensor fusion task. Moreover, due to the fixed priority-based scheduling policy, the increased execution time of the configurable sensor fusion task will impact all the subsequent tasks in the ready queue with lower priorities. For instance, as shown in Figure 3, the runtime execution time increase of the configurable sensor fusion task will also impact object tracking and image pre-processing control cycles in the following control cycles, which leads to a continuously high deadline miss ratio. Moreover, if the computation of the configurable sensor fusion cannot be completed within the deadline, the fusion results of this control cycle are discarded and the subsequent tasks cannot receive the results of the obstacle prediction analysis. Therefore, the vehicle cannot update its speed in a timely manner at this point resulting in poor tracking performance. Due to the high deadline miss ratio, the vehicle speed update becomes sluggish. Figure 4(b) shows the speed difference between the two vehicles. At t=23.4s when the speed of car B reduces to 2m/s before it fully stops, a collision occurs between the two cars (the distance between the two cars becomes 0). In summary, the car following application performs poorly because the scheduling based on fixed priority can not cope with the dynamic behavior of the tasks, resulting in continuous deadline misses and possible accidents.



(a) Scheduling output with traditional adaptive scheme.

(b) Desired scheduling output.

Fig. 5. Scheduling output with traditional adaptive scheduling scheme and preferred scheduling output for autonomous driving.

On the other hand, the existing adaptive scheduling approach cannot perform well in autonomous driving scenarios either. Figure 5 shows an example where t_i-j represents task i's release in control cycle j. We can see that there are three tasks (i.e., t_1, t_2, t_3) released in each control cycle. The control command can be generated only when all the three tasks can be completed before the corresponding deadlines. We assume that execution time of each task is 1s and deadlines of the corresponding tasks are as follows: t_1-1 : $1s$, t_1-2 : $4s$, t_1-3 : $7s$, t_2-1 : $8s$, t_2-2 : $9s$, t_2-3 : $10s$, t_3-1 : $11s$, t_3-2 : $12s$, t_3-3 : $13s$. Figure 5(a) shows the scheduling output according to the adaptive scheduling scheme which keeps monitoring the schedulability of the system and adaptively schedules the corresponding tasks in order to guarantee that the tasks can

meet their deadlines. We can see that all the tasks can meet their deadlines and the control commands are generated at t=7s, 8s and 9s respectively. However, under certain driving scenarios (e.g., the front car suddenly brakes), the control commands are expected to generate as soon as possible to get a responsive reaction. Thus, the scheduling output as shown in Figure 5(b) is preferred, which generates the control commands earlier (i.e., at t=3s, 6s, and 9s). In summary, the existing adaptive scheduling methodology is also not efficient to autonomous driving for the reason that it does not take the driving performance into account and cannot generate control commands at the right point. Thus, a framework that can intelligently coordinate autonomous driving tasks with high execution time variation through well balancing the real-time, responsiveness and throughput according to the driving performance is critically required.

III. System Design

A. Problem Formulation

The system considered in this paper consists of a set Γ of n real-time tasks. These tasks are released periodically and are executed on a multiprocessor platform composed of M identical processors $p_1 ..., p_M$. The dependencies among the tasks are modeled as a directed acyclic graph (DAG) as shown in Figure 2. The nodes in the graph represent the tasks that are connected by the corresponding edges. Edge $e_{i,j} \in E$ connects task τ_i and task τ_j and defines a precedence constraint between the two tasks. For each task, the set of immediate predecessors is defined as $ipred(\tau_i) = \{\tau_j \in \Gamma : \exists e_{j,i} \in E\}$. Task τ_i can be released and pushed into the ready queue only after the completion of all its immediate predecessors $ipred(\tau_i)$. In addition, task τ_i must be completed within D_i units of time after its release. Otherwise, the output of task τ_i will be discarded. In addition, the tasks with higher priority will be executed first in a non-preemptive manner within the ready queue. A task without incoming edges is referred to as a source task, whereas a task without outgoing edges is denoted as a sink task. In an autonomous driving system, the source tasks usually refer to the sensing-related tasks (e.g., image pre-processing, point cloud pre-processing) whose task rates can be configured in an allowable range. For instance, in this work, the allowable range of GPS (IMU) is $[10hz, 100hz]$. The sink tasks usually refer to the control tasks which generate control commands (e.g., steering throttle, brake) to the vehicle.

In this work, HCPerf is designed to effectively utilize the system resources to guarantee the end-to-end deadline while dynamically prioritizing the responsiveness or throughput of control commands according to the driving performance of the autonomous driving vehicle. Specifically, the problem can be expressed as follows: given a performance target R of an autonomous driving control application, and the allowable rate range of the source tasks $[r_i^{min}, r_i^{max}]$, dynamically choose the task rate r_i and the priority adjustment coefficient γ (a coefficient that well balances the deadline and priority of each task according to the vehicle driving performance) such that in the k^{th} control period, the tracking error $E(k)$ between the performance target $R(k)$ and the actual performance of the autonomous vehicle $P(k)$ is minimized. The problem can be formulated as Eq.(1a).

$$\min_{\gamma(k), r_i(k)} |R(k) - P(k)| \tag{1a}$$

$$\text{s.t. } \gamma^{min} \leq \gamma(k) \leq \gamma^{max} \tag{1b}$$

$$r_i^{min} \leq r_i(k) \leq r_i^{max} \tag{1c}$$

Constraint (1b) ensures that the scheduling policy γ is within the scheduling space. Constraint (1c) ensures that rates of all the tasks are within the corresponding allowable ranges. Moreover, it should be noted that $P(k)$ and $R(k)$ are application-dependent. For instance, in the car following example, the performance target $R(k)$ is configured as the speed of the lead car, while the performance of the autonomous driving car $P(k)$ is the real speed in each control cycle.

B. System Overview

Figure 6 shows the architecture and workflow of HCPerf which mainly consists of two coordinators in order to intelligently schedule the autonomous driving tasks.

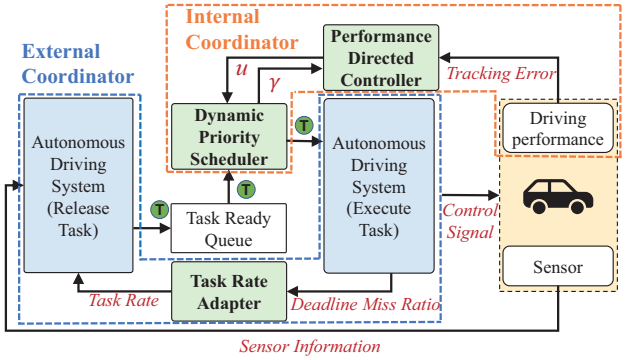


Fig. 6. System architecture and workflow of HCPerf.

Internal Coordinator. The internal coordinator mainly contains the following two components: the Performance Directed Controller and the Dynamic Priority Scheduler. It adaptively schedules the tasks according to the driving performance of the vehicle. In each control period, the autonomous vehicle first obtains information about its surrounding environment through the sensors (e.g., camera for the car following case). Then the on-board system calculates the tracking error (e.g., difference between the speed of the lead car and the following car in car following). The Performance Directed Controller uses the tracking error to estimate the nominal priority adjustment coefficient u. Taking u as input, the Dynamic Priority Scheduler determines the $actual$ priority adjustment coefficient γ and then adjusts the priorities of the ready tasks, which are then dispatched based on the adjusted priorities. The estimation process for u intelligently considers task deadlines and vehicle driving performance in order to achieve the best balance. Finally, tasks are executed and control commands for the vehicle are produced. The process then iterates.

External Coordinator. Although the internal coordinator can intelligently schedule the tasks within the ready queue according to the driving performance of the vehicle, inefficiency still exists in the system due to the large execution time variance of autonomous driving tasks. First, when the system is overloaded, deadline misses of the end-to-end tasks can frequently occur which prevents generating control commands and also wastes system computing resources. Second, when the system is underloaded, the system is not fully utilized. In this case, more tasks can be released to improve the throughput of control inputs to achieve smoother control. The external coordinator is designed to improve the system efficiency in a feedback manner. Specifically, a Task Rate Adapter is designed in the external coordinator to adaptively adjust the rate of the source tasks in order to 1) efficiently utilize the system resources to improve the throughput of control commands to the vehicle and 2) minimize the end-to-end deadline miss ratio.

In the following sections, we discuss the design of the major components in detail. Table I shows the summary of notation symbols utilized in the following sections.

TABLE I
SYMBOL DESCRIPTION.

Symbol	Description
c_i	Execution time of task i.
D_i	Relative deadline of task i.
p_i	Priority of task i configured by the system (Smaller value means higher priority).
d_i	Scheduling deadline of task i.
P_i	Dynamic scheduling priority of task i (Smaller value means higher priority).
n_p	Number of processors.
T_p	Remaining processing time of task on processor p.
u	Nominal priority adjustment coefficient.
Q	The task set in ready queue.
γ	Actual priority adjustment coefficient.
E	Vehicle performance tracking error.
r_i	Rate (i.e., frequency) of task i.
$m(k)$	Deadline miss ratio of the system in k-th period.
F	Model-free control offset term.
α	Constant control gain of model-free control.
K	Feedback gain of Model-Free Control.
K_p	Control gain of Task Rate Adapter.
$\mathbf{r}$	The vector of task rate.

IV. PERFORMANCE DIRECTED CONTROLLER

In this section, we present the Performance Directed Controller, which is responsible for reducing the tracking error $E(t)$ by regulating the nominal priority adjust parameter $u(t)$ which is used to determine the priority adjustment coefficient γ (see Section V for details), which in turn determines the priority of tasks in the ready queue. Thus *scheduling is directed by realtime performance of the autonomous vehicle*. Since the explicit mathematical relationship between $E(t)$ and $u(t)$ is unknown, we design the Performance Directed Controller based on the Model-Free Control (MFC) in [17] to fulfill this goal.

MFC is a data-driven and learning-free control algorithm. It approximates the complex and probably time-varying relationship between the car tracking error $E(t)$ and the nominal priority adjustment parameter $u(t)$ as a first-order linear system, as:

$$\dot{E}(t) = F(t) + \alpha u(t). \tag{2}$$

where $\dot{E}(t)$ represents the derivative of the car tracking error $E(t)$. $F(t)$ is an offset term condensing both the unmodeled system dynamics and the external disturbance. $F(t)$ is continuously updated online to guarantee the accuracy of Eq.(2) within a short period of time. Note that this idea is similar to approximating a complex function by a piecewise constant function. $\alpha < 0$ is the constant control gain.

With Eq.(2), we can formulate a classical control problem as: *Design the command $u(t)$, such that the system output $E(t)$ can accurately follow its reference value $E(t)^* = 0$.*

Assume that the offset term $F(t)$ in Eq.(2) has been estimated as $\hat{F}(t)$, then we can formulate the command $u(t)$ as:

$$u(t) = \frac{-\hat{F}(t) + KE(t)}{\alpha} \tag{3}$$

with $K < 0$ as the feedback gain.

Substituting Eq.(3) back into Eq.(2) yields:

$$\dot{E}(t) = KE(t) + F(t) - \hat{F}(t), K < 0. \tag{4}$$

With $\hat{F}(t)$ close to $F(t)$, the tracking error $E(t)$ will stay inside a bounded ball around the origin [18].

By re-arranging Eq.(2), we derive $\hat{F}(t)$ in Eq.(3), as:

$$\hat{F}(t) = \hat{\dot{E}}(t) - \alpha u\left(t - T_s\right), \tag{5}$$

with $u\left(t - T_s\right)$ as the nominal priority adjustment parameter at the last sampling step and T_s as the control sampling period of MFC. Consequently, the estimation of the offset term $\hat{F}(t)$ boils down to the problem of estimating $\dot{E}\left(t\right)$.

Directly differentiating the measured car tracking error $E(t)$ will amplify the measurement noise. Instead, we adopt the Algebraic Differentiation Estimation (ADE) [19] to estimate $\dot{E}(t)$.

ADE approximates $\dot{E}(t)$, the first-order *derivative* of $E(t)$, as its time-weighted *integral* [20]:

$$\hat{\dot{E}}(t) = \frac{6}{T_{ADE}^3} \int_0^{T_{ADE}} \left(T_{ADE} - 2\tau\right) E(t-\tau) d\tau. \tag{6}$$

In Eq.(6), T_{ADE} is the width of a sliding window, inside which the measured value of $E(\tau), \tau \in [t - T_{ADE}, t]$ is recorded. Note that the integral operator in Eq.(6) serves as a low-pass filter to attenuate the high frequency measurement noise.

Substituting Eq.(6) into Eq.(5) yields $\hat{F}\left(t\right)$. Substituting Eq.(5) into Eq.(3) finally leads to $u(t)$.

Remark: Substituting Eq.(5) back into Eq.(3) yields:

$$u(t) - u(t - T_s) = -\frac{\hat{\dot{E}}(t)}{\alpha} + \frac{KE(t)}{\alpha} \tag{7}$$

Dividing both sides of Eq.(7) with T_s leads to:

$$\dot{u}(t) = -\frac{\hat{\dot{E}}(t)}{\alpha T_s} + \frac{KE(t)}{\alpha T_s} \tag{8}$$

As will be shown in the experimental results in Section VII-B, $|\dot{E}(t)| \ll |E(t)|$. With $K = -1$, the sign of $\dot{u}(t)$ depends principally on the sign of $E(t)$. Therefore, the nominal scheduling control parameter $u(t)$ serves as two purposes:

- When the car tracking error $E(t)$ becomes large, $\dot{u}(t)$ and consequently $u(t)$, will increase ($\alpha < 0$), so as to prioritize the vehicle control task and endeavor to reduce $E(t)$ in order to achieve responsive control.
- Instead, if $E(t)$ remains reasonably small, $u(t)$ will remain stable. In this case, the task scheduler will prioritize other tasks (e.g., tasks with the earliest deadlines) of the autonomous driving system and guarantees the overall task throughput.

The *nominal* priority adjustment parameter $u(t)$ serves as an input to the Dynamic Priority Scheduler, which produces the *actual* priority adjustment parameter γ by upper and lower-bounding $u(t)$. The details can be found in Section V.

V. Dynamic Priority Scheduler

The Dynamic Priority Scheduler intelligently schedules the tasks through jointly balancing the priority, the deadline and runtime performance of the autonomous vehicle. Figure 7 shows the workflow of the Dynamic Priority Scheduler.

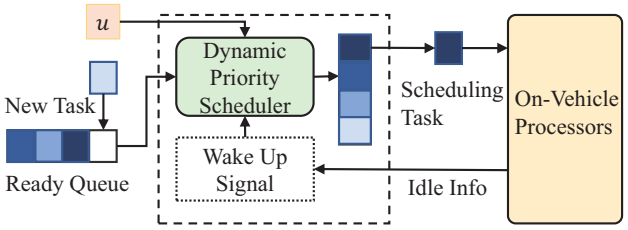


Fig. 7. Workflow of the Dynamic Priority Scheduler.

In an autonomous driving system, different tasks have different execution times. Thus, the relative deadline D_i is not an effective metric to determine whether a task is more urgent to run in order to meet the deadline d_i. For instance, a task with loose relative deadline and longer execution time can be more urgent to be scheduled than a task with tight relative deadline but shorter execution time. Thus, we define the metric, scheduling deadline, which represents the maximum delay between the tasks' release time and the start of its execution in order to complete before the deadline as follows:

$$d_i = D_i - c_i \tag{9}$$

where D_i denotes the relative deadline of task τ_i, and c_i represents the execution time of τ_i observed by the system.

In order to jointly balance the impact of scheduling deadline, priority and performance of the vehicle, we define the *dynamic scheduling priority* P_i of a task τ_i as follows:

$$P_i = \gamma * p_i + d_i \tag{10}$$

where the *priority adjustment coefficient* $\gamma \in \mathbb{R}^{0+}$ is utilized to balance the weight between the scheduling deadline d_i and the initial priority p_i. Specifically, when the value of γ is small, the scheduling approximates the deadline-based algorithm which guarantees the completion of tasks with earliest deadlines [21]. On the other hand, when the value of γ is large, the scheduling approximates the priority-based algorithm which ensures the completion of important tasks. The Dynamic Priority Scheduler maintains a list for recording the scheduling priority P_i of each task. Whenever a new task is released and pushed into the ready queue, it is added to the list as well. Hence the coefficient γ determines the magnitude of the P_i value for each task.

Deriving γ from $u(t)$. We determine the value γ from the nominal priority adjustment parameter $u(t)$ in order to intelligently consider the impacted driving performance (e.g., tracking error). Specifically, to guarantee that all the tasks in the ready queue Q can be completed before the corresponding deadlines, we first determine the allowable range of γ, $[0, \gamma_{max}]$ that satisfies the following constraints:

$$\begin{cases} c_1 + \frac{\sum T_p}{n_p} + \frac{\sum_{P_i<P_1} c_i}{n_p} < D_1, & i \in Q \\ c_2 + \frac{\sum T_p}{n_p} + \frac{\sum_{P_i<P_2} c_i}{n_p} < D_2, & i \in Q \\ \cdots \end{cases} \tag{11}$$

where the first term in each equation (e.g., c_1) represents the execution time of the task observed by the system, i.e. the execution time from the last run of the task. The second term, $\frac{\sum T_p}{n_p}$, denotes the parallel processing time required to complete the existing tasks currently running on the processors. The third term (e.g., $\frac{\sum_{P_i<P_1} c_i}{n}$) denotes the time required to run those tasks that have higher dynamic priority in the processor pool.

We can find that there are two possible outcomes in the search for γ_{max}:

1) No γ_{max} exists in the allowable range that can meet the constraints in Eq.(11). This means that no scheduling solution exists that can guarantee that all the tasks meet their deadlines. This is due to the system is overloaded and cannot handle the tasks. This situation can be detected and adjusted with the external coordinator which reduces the task rates of the corresponding tasks. In this case we set γ to zero.

2) A feasible γ_{max} can be obtained. In this case, the nominal priority adjustment parameter $u(t)$ from the Performance Directed Controller is used to determine γ. Note that $u(t)$ from Eq.(3) does not consider the bounds for γ which are derived from the need to guarantee the deadline for all waiting tasks at the current moment. Therefore, we need to pick the best matching value for u within the range $[0, \gamma_{max}]$. The scheduler maps u to γ in the range $[0, \gamma_{max}]$ as follows.

$$\gamma = \begin{cases} 0, & u(t) < 0 \\ u(t), & 0 \le u(t) \le \gamma_{max} \\ \gamma_{max}, & u(t) > \gamma_{max} \end{cases} \tag{12}$$

Finally, the scheduler uses the computed γ to sort the task queue. At the time when a task can be dispatched, all the tasks in the ready queue will have their dynamic priorities computed according to Eq.(10) and the one with the highest dynamic scheduling priority is dispatched.

VI. Task Rate Adapter

The Task Rate Adapter is designed as a feedback controller. It takes the deadline miss ratio of the tasks as input and regulates the rates of the source tasks in order to effectively utilize the system resources. Specifically, the Task Rate Adapter adaptively tunes the system load in order to 1) fully utilize the system resources to improve the throughput of the control commands and 2) minimize the deadline miss in order to avoid wasting computing resources.

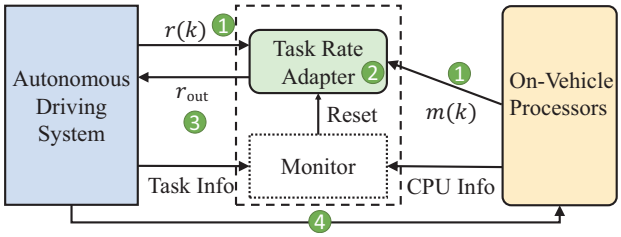


Fig. 8. Workflow of Task Rate Adapter.

Figure 8 shows the architecture and workflow of the Task Rate Adapter. It is worth noting that the Task Rate Adapter jointly considers all the processors and adjusts the rates of all the source tasks at the same time due to the following reasons.

1) Unlike the conventional vehicle system where each task is bounded to a specific processor, our system does not bound each task to a particular processor *a priori*. Instead, according to the variation of task execution time and task scheduling algorithms, the same tasks released in different periods can be executed on different processors in order to fully utilize the system resource. Thus, in this work, we consider the deadline misses on all the processors collectively.

2) End-to-end tasks existing in an autonomous driving system should be considered comprehensively. HCPerf is designed to improve the performance of the whole system by jointly considering the rates of all the existing tasks at the same time. For instance, a task can be triggered or executed only when all its precursor tasks are completed. Thus, the least waiting time can be achieved if all its precursor tasks can be completed at the same time by jointly tuning the rates of them in a comprehensive manner.

An autonomous driving system has a stringent real-time requirement. Thus, a proportional controller, which incurs low runtime overhead, is designed for online task rate adaptation.

Let $R = \{r_1, r_2, ..., r_N\}$ be the set of adjustable task rates in the autonomous driving system. As shown in Figure 8, the workflow of the Task Rate Adapter is as follows:

(1) Given a target deadline miss ratio m_t and the measured deadline miss ratio of the system $m(k) \in \mathbb{R}$, where k denotes the control period index, the deadline miss ratio error is computed as $e(k) = m_t - m(k)$. If $m(k) = 0$, $e(k)$ is set to a pre-defined small positive value.

(2) Based on $e(k)$ and the offline profiled data, the Task Rate Adapter computes a control input $\mathbf{r_{out}} \in \mathbb{R}^N$ to optimize the deadline miss ratio target,

$$\mathbf{r_{out}} = K_p e(k) + \mathbf{r}(k), \tag{13}$$

where $\mathbf{r}$(k) represents the current system task rates, $\mathbf{r_{out}}$ indicates the adapted task generation rates, and $K_p \geq 0$ is the control gain. At system initialization, K_p is set from offline profiled data. If $e(k) < 0$, the system is considered overloaded and the task rate adapter reduces the task generation rates to alleviate the system load. On the other hand, if $e(k) > 0$, the task rate adapter increases the task generation rates to enhance the system-level task execution throughput. The Task Rate Adapter will gradually decrease the value of K_p to reduce the influence of $e(k)$ as the system becomes more and more stable. When K_p is reduced to 0, the task rates become stable. However, when the Task Rate Adapter observes an unusual change in the system's task processing time variations, K_p will be reset to the value from offline profiled data.

(3) The autonomous driving system applies the configuration $\mathbf{r_{out}}$ in order to conduct the end-to-end tasks.

In summary, the Task Rate Adapter can quickly adjust the system load to effectively solve the problem of overload or underload which cannot be effectively solved with the internal coordinator. In this way, the system can meet the maximum throughput without deadline miss, and the system resource can be effectively utilized. At the same time, the Task Rate Adapter can help to guarantee the schedulability of the system through maintaining the utilization of the processors below the specified utilization bound according to [21].

Stability Analysis: Here we analyze the stability of our external coordinator when the estimated deadline miss ratio is different from the real deadline miss ratio at runtime. $g \geq 0$ represents the proportion of the task rate to the system load to the change of deadline miss ratio. The deadline miss ratio $m(k+1)$ can be expressed as:

$$m(k+1) = m(k) + g * (r(k+1) - r(k)) \tag{14}$$

When $k \to +\infty$ which means the system has been running for a sufficiently long time, the system task rate $r(k+1) = r(k)$ due to the feedback regulation of the Task Rate Adapter. Thus Eq.(14) becomes $m(k+1) = m(k)$, so the curve of deadline miss ratio will be smooth. This analysis shows that our external coordinator can effectively handle the task rate and minimize the deadline misses ratio to a stable situation.

It is worth noting that, HCPerf mainly focuses on the coordination of the tasks that utilize CPU. Although the tasks such as object detection use GPU at the same time during the detection process, they also use CPU for data fetching. For the GPU, HCPerf dynamically monitors and records its execution time and tries to guarantee the end-to-end deadline.

VII. Evaluation

A. Experimental Setup

1) **Simulation Testbed:** We build the simulator based on Apollo [1]. Figure 9 shows the corresponding architecture which consists of: 1) an Auto-Driving Simulator, 2) a Vehicle

Control Simulator and 3) the Scheduling Framework. Specifically, the Auto-Driving Simulator is implemented as a distributed real-time system to simulate the execution of the tasks with different dependencies, the corresponding communication and resource allocation. The Vehicle Control Simulator simulates the trajectories of an autonomous vehicle. When it receives control commands from the Auto-Driving Simulator, it directs the vehicle to perform corresponding actions such as acceleration, deceleration, and steering. The Scheduling Framework mainly encapsulates different scheduling schemes.

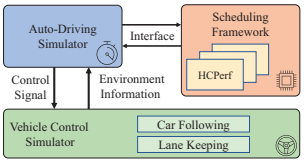


Fig. 9. Simulation Testbed.

Fig. 10. Hardware Testbed.

2) **Hardware Testbed:** Our hardware testbed consists of 1:10 scaled cars as shown in Figure 10. Each scaled car is built on a realistic one-tenth scaled car chassis with a four-wheel-drive brushless drivetrain for fast acceleration and steering. Moreover, the car is equipped with a variety of sensors, including a 2D scanning LiDAR as the primary navigation sensor, an inertial measurement unit (IMU), and a camera. The sensing and actuation components are controlled using an onboard embedded computer (Core-3399J) [22]. We incorporate the scheduling and control schemes into the system to interact with the onboard system for evaluation.

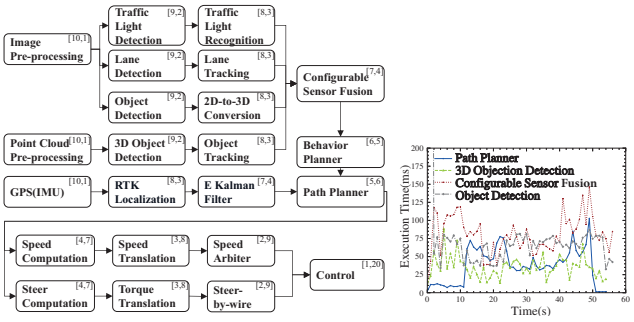


Fig. 11. Autonomous driving tasks and their dependencies. [Priority, Value] represents the priority and value of the task.

Fig. 12. Example of the execution time of autonomous driving tasks.

3) **Task Graph:** Figure 11 shows the task graph adopted in the evaluation which includes 23 tasks from sensing to control. For each task, we also present its priority-value pair in brackets. The smaller the priority number is, the higher its priority and the higher its value. Moreover, we measure the execution time of the tasks in different environments. Figure 12 shows the results for four of them as an example. Based on the measurement, we determine the range of execution time for each task. All the measurements are obtained by running Apollo [1] on the Nvidia Jetson TX2 [23] platform which is widely adopted in autonomous driving research [24].

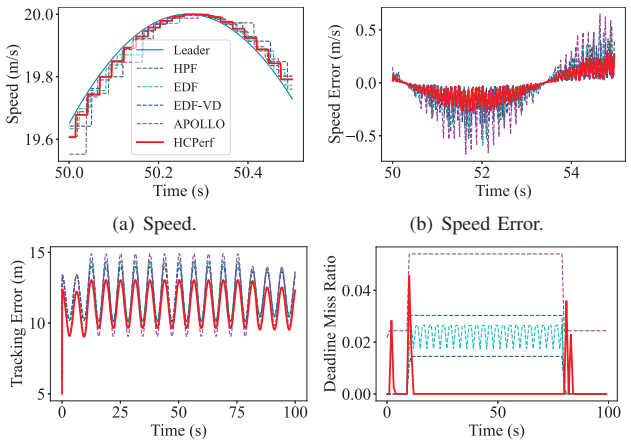
4) **Baselines:**

- HPF [25]: HPF (High Priority First) assigns a priority to each task offline. The task with the highest priority is scheduled to be executed in a non-preemptive way.
- EDF [21]: EDF (Earliest Deadline First) assigns priority to tasks based on their absolute deadlines. The task with the earliest deadline gets the highest priority.
- EDF-VD [8]: In EDF-VD, the deadlines of all high-criticality tasks are shortened with a scaling factor and this modified deadline is called the virtual deadline. At runtime, all high-criticality tasks are executed according to their virtual deadlines and all low-criticality tasks are executed with their actual deadlines using EDF.
- Apollo [1]: Apollo is the state-of-the-practice. It binds different tasks to different processors and then uses the statically assigned priority to select tasks for execution.

B. Evaluation with Different Applications

1) Evaluation with car following:

We first evaluate the effectiveness of HCPerf with car following. The performance metric is configured as the tracking error between the following car and the lead car [14]. The following driving case is utilized: the speed of the lead vehicle follows a sine function with a period of 7s and the speed is bounded in the range $[10m/s, 20m/s]$. In addition, at t=10s, we increase the execution time of the configurable sensor fusion task from 20ms to 40ms to emulate a load increase as the result of encountering a complex scene. The execution time stays at the high level until t=80s when it is restored to the normal value. The task execution time and corresponding deadlines are configured according to the discussion in Section VII-A and the work presented in [24], [26].



(a) Speed. (b) Speed Error.

(c) Distance Tracking Error. (d) Deadline Miss Ratio.

Fig. 13. System performance of the scenario in which the speed of the lead car follows a sine function.

Figure 13 represents the evaluation result. In particular, Figure 13(a) shows the speed of the lead vehicle and that of the following vehicle with different schemes in a 0.5s time window. Specifically, a small time window is selected to

highlight the difference for easy visualization. We can see the speed of the following car with HCPerf is the closest to that of the lead car. Figure 13(b) shows the corresponding speed error (speed of the lead car v_r minus that of the following car v). Table II summarizes the RMS of the speed tracking errors. HCPerf has the best performance here with an RMS of $0.55m/s$ while the other schemes have much higher speed errors due to the following reasons. When the tracking error begins to increase, HCPerf gives higher priority to the control related tasks to generate control commands as soon as possible and make the time interval between two control commands as short as possible. In this case, the speed of the following vehicle can be frequently updated to achieve better driving performance. However, the fixed priority-based scheduling scheme adopted in Apollo cannot make the update in time without taking the tracking error or the execution time variance into account. HPF allocates more computing resources to the pre-defined important tasks. Thus, the other tasks usually miss their deadlines and the control commands cannot be effectively generated in this case which leads to low performance. For EDF and EDF-VD, they first schedule the tasks that have the earliest deadlines. Thus, the control task cannot be timely scheduled. In this case, the control commands cannot be timely generated which leads to a delay in speed updates.

TABLE II
ROOT MEAN SQUARE VALUES OF SPEED TRACKING ERROR.

	HPF	EDF	EDF-VD	Apollo	**HCPerf**
RMS (m/s)	1.02	0.99	0.78	1.28	**0.55**

Figure 13(c) represents the distance error of the following car which indicates the distance between the two cars with different schemes. Since the speed of the lead car oscillates in this experiment, we see that the distance errors for all the schemes also oscillate. What is important here is the magnitude of the oscillation. HCPerf shows the least amount of fluctuation. The RMS values of distance tracking error are summarized in Table III.

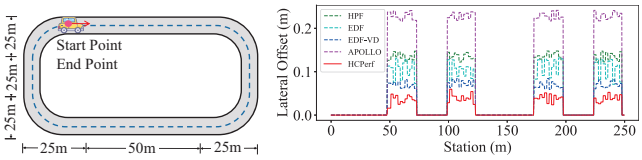
TABLE III
ROOT MEAN SQUARE VALUES OF DISTANCE TRACKING ERROR.

	HPF	EDF	EDF-VD	Apollo	**HCPerf**
RMS (m)	12.24	12.22	12.07	12.31	**11.27**

Figure 13(d) shows the deadline miss ratio during this process. At first, all the schemes can meet the task deadlines due to the very low system load. HCPerf has a deadline miss at t=0s due to the need for the Task Rate Adapter to reset when the system has an important change or when the system is just being initialized. The controller needs a brief period of adjustment to find the optimal task rate to determine the best load for the system. At t=10s, due to the increase in the task execution time, the baseline schemes start to generate deadline misses, and this situation lasts until t=80s when the low system load is restored. In contrast, HCPerf detects the increase in system load and dynamically adjusts the task rates accordingly. As a result, it only experiences a brief period where deadline misses occur and then quickly brings the deadline miss ratio to zero and keeps it at that level until t=80s when the load sees another big change. At t=80s when the system load decreases, the deadline miss ratio of all the schemes decreases. HCPerf, similar to what happened at t=10s, quickly brings the deadline miss ratio down to 0 after a brief adjustment.

2) Evaluation with Lane Keeping:

We evaluate the effectiveness of HCPerf with the other representative application, lane keeping. Specifically, lane keeping enables the autonomous driving vehicle to follow the desired lane by adjusting the front steering angle [27]. Performance metric is configured as the lateral offset position of the vehicle from the centerline of the planned path in this case. For the setup, we configure the longitudinal speed of the vehicle to be fixed at $5m/s$ and the objective of the controller is to minimize the lateral offset of the vehicle from the center of the lane.



(a) Loop Driving. (b) Lane Keeping Offset.

Fig. 14. Simulation experiment result of loop driving.

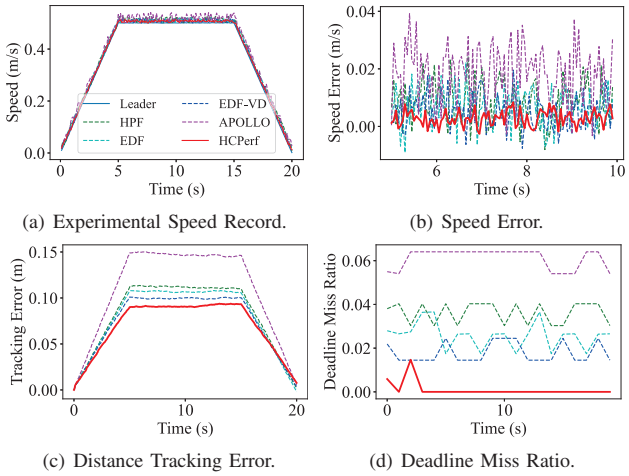
Figure 14(a) shows the loop driving scenario in which the autonomous car drives along an oval-shaped closed-loop following the clockwise direction. The performance is measured by the deviation of the car from the center of the lane. The smaller the deviation is, the better the performance. Figure 14(b) shows the result of lane-keeping errors (i.e. lateral offsets). When the car is driving in a straight line, the lane-keeping offset errors are 0. However, during the four turns in the loop, we can see that the difference in the performance among different schemes is prominent. HCPerf has much better performance due to the following reasons. Specifically, when the vehicle is driving in a straight line without any steering operation, the detected tracking error is 0. In this case, HCPerf schedules the tasks in order to maximize the throughput and tries to guarantee that all the tasks in the ready queue can meet their deadlines. However, when the vehicle starts to turn and the lane offset (tracking error) is observed, HCPerf dynamically adjusts the priority in order to generate the control commands in a short time. Then, the system generates the steering operation quickly to reduce the error of lane-keeping offset. Unlike HCPerf, Apollo conducts scheduling only based on each task's fixed priority, leading to the worst lane-keeping performance. Other baselines are unaware of the vehicle lane offset, so they keep the original scheduling method, resulting in large errors. This is because the control commands cannot be generated fast enough even if they ensure the tasks can meet their deadlines. The RMS values of lateral offset errors are summarized in Table IV.

TABLE IV
ROOT MEAN SQUARE VALUES OF LATERAL OFFSET ERROR.

	HPF	EDF	EDF-VD	Apollo	**HCPerf**
RMS (m)	0.093	0.075	0.051	0.159	**0.027**

3) Hardware Evaluation:

In this section, we evaluate the effectiveness of HCPerf with the car following application using the hardware testbed. Two scaled cars are used in the experiment: the lead car is set to follow a fixed driving routine, while the following car is set to car following mode. Each test runs for 20s. In the experiment, the lead car first performs an acceleration for 5s, and then maintains a constant speed for 10s, followed by a deceleration for 5s. The cars run in a straight line without turning.



(a) Experimental Speed Record. (b) Speed Error.

(c) Distance Tracking Error. (d) Deadline Miss Ratio.

Fig. 15. Hardware experiment with car following.

Figure 15 shows the evaluation results. Figure 15(a) shows the overall speed record of the vehicles in the car following process. In general, the autonomous driving vehicle follows the reference vehicle through three stages of acceleration, smooth running and deceleration. Figure 15(b) shows the speed error of the following car from 5s to 10s. We can see that HCPerf not only has the smallest speed error, but also has the smallest magnitude of fluctuation. Table V summarizes the speed tracking error of different schemes.

TABLE V
ROOT MEAN SQUARE VALUES OF SPEED TRACKING ERROR.

	HPF	EDF	EDF-VD	Apollo	**HCPerf**
RMS (m/s)	0.015	0.013	0.012	0.021	**0.009**

Figure 15(c) shows the distance tracking error, which represents the variation of the distance between the following car and the lead car in the 20s period. Unlike the simulation, the speed record of the lead car is affected by the presence of noise. Thus, if the following car cannot responsively generate the control commands, it will lead to more serious tracking errors. Thus, HCPerf can significantly decrease the distance tracking error by 46.2% compared with Apollo and 29.2% to the baselines on average. In particular, the speed error appears below 0 due to the speed noise and the lag in the throttle control of the scaled car can be observed. The RMS values of distance tracking errors are summarized in Table VI.

TABLE VI
ROOT MEAN SQUARE VALUES OF DISTANCE TRACKING ERROR.

	HPF	EDF	EDF-VD	Apollo	**HCPerf**
RMS (m)	0.084	0.083	0.072	0.117	**0.063**

Figure 15(d) shows the deadline miss ratio in the hardware experiment. We record the result once per second. After the initial adjustment procedure, HCPerf is able to keep the deadline miss ratio at 0 for the rest of the experiment, while other schedulers have deadline misses throughout the whole car following process in the range of 2-6%.

C. Prioritize Responsiveness and Throughput

In this section, we use another real-world example to show how HCPerf effectively prioritizes the responsiveness and throughput of the control commands in different scenarios according to the driving performance. Specifically, we use the response time of control commands to quantify the responsiveness of the system which is defined as the duration between the release and execution of the control task. Moreover, as the throughput directly impacts the passenger perceived experience, we adopt the passenger perceived comfort [5] to dictate the system throughput. The higher the throughput is, the more abrupt acceleration and deceleration can be avoided. Thus, better passenger perceived comfort can be achieved.

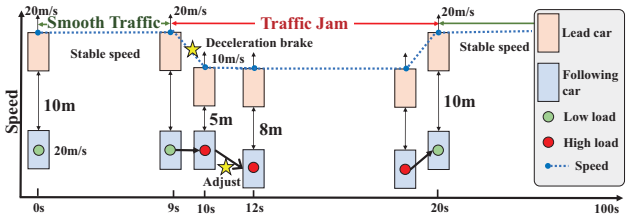
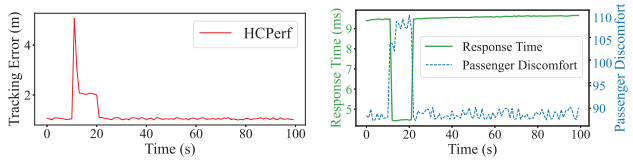


Fig. 16. The overall driving process of the car following application.

Figure 16 shows the overall driving process of the car following application. At the beginning, the two cars drive at a stable speed with $20m/s$. When T=10s, the lead car starts to decelerate facing a traffic jam. As the number of vehicles in the surrounding grows, the completion time of corresponding tasks increases which leads to deadline misses and the control commands cannot be generated in time. At the same time, the distance between the two vehicles starts to decrease and the tracking error increases accordingly. When the tracking error is detected, HCPerf starts to improve the responsiveness of the control commands in order to avoid potential accidents and mitigate the tracking error.

Figure 17 represents the tracking error, the response time of the control command and the user-perceived comfort.

As shown in Figure 17(a), the tracking error dramatically increases to 5m when t=10s. According to the tracking error, HCPerf allocates computing resources to the control task in order to generate as many as possible control commands immediately in order to achieve responsive control to avoid a potential collision. Thus, the tracking error is effectively mitigated to 2m when t=12s. Figure 17(b) shows that the response time of the control task is reduced and the passenger discomfort increases in this process. This is the reason that the computing resources are allocated to the control tasks and the other tasks with tighter deadlines may miss their deadlines. Thus, the overall system throughput decreases during this period. After t=20s, when the two vehicles pass by the road sections with traffic jam, the number of surrounding vehicles decreases and the system load decreases accordingly. Thus, the tracking error is further mitigated. Then, HCPerf allocates more computing resources to the tasks with tighter deadlines to improve the system throughput. After T=20s, the passenger discomfort is effectively reduced to provide a good experience.

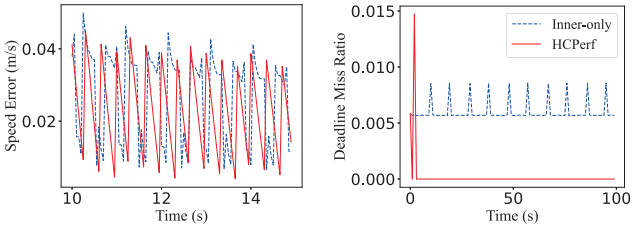


(a) Tracking Error. (b) RT vs. Passenger Discomfort.

Fig. 17. Tracking error, response time and passenger discomfort.

D. Ablation Study

In this section, we conduct an ablation analysis by comparing the full version of HCPerf with a version that does not include the External Coordinator. The experiment setup has referenced the car following simulation setup.



(a) Speed Error. (b) Deadline Miss Ratio.

Fig. 18. System performance with and without the External Coordinator.

From figure 18(a), we can see that the full version has smaller fluctuations in speed tracking, which means better driving performance. Figure 18(b) shows that, due to the absence of the External Coordinator, the system produces a low deadline miss ratio throughout the simulation that cannot be reduced to 0 as the full version does, which leads to a failure to guarantee the task deadlines. In terms of the final distance tracking error, the full version is 0.5m better than the Internal Coordinator version. In summary, the Internal Coordinator of HCPerf can optimize the driving performance, while the External Coordinator can regulate the deadline miss ratio, and the two parts cooperate to obtain better performance.

E. Overhead Analysis

We use the hardware platform to measure the computation overhead of HCPerf. The analysis is the average overhead for multiple scenarios. The methods utilized in the paper have linear complexity, so there are no huge fluctuations in computational overhead. The overhead consists of the computation overhead for Internal Coordinator and External Coordinator. The measured average execution time of HCPerf is less than 5ms per period of 1s, which is moderate for implementation.

VIII. Related Work

Resource Management in Autonomous Driving Platform. Autonomous driving has gained significant interest across industry and academia, leading to numerous studies aimed at optimizing accuracy, efficiency, and real-time performance. Bateni et al. [28] presented a temporally predictable energy optimization framework for running DNN workloads in GPU-enabled automotive systems. Bateni et al. [29] developed a model to predict the system overhead of each memory management method based on application characteristics, and by performing per-task memory management policy switching and kernel overlap, the scheduler can relieve system memory pressure and reduce multitasking response time. However, these approaches mainly focused on the trade-off between real-time and accuracy in perception, while HCPerf considers end-to-end real-time performance across sensing, perception, prediction, planning, and control.

Real-time Scheduling. Numerous studies propose scheduling and analysis methodologies to ensure timing correctness for embedded systems. Guo et al. [6] proposed a cache-aware partitioned scheduling that covers inter-task cache interference. Choi et al. [7] proposed a class-level fixed priority preemptive scheduler and its schedulability analysis framework with weak hard real-time constraints. Liu et al. [8] studied real-time scheduling of hybrid critical systems, where low-critical tasks can still guarantee some services in high-critical mode and reduce the execution budget. Bateni et al. [9] proposed a time-predictable runtime system capable of guaranteeing DNN workload deadlines by efficient approximation. Existing task scheduling and analysis mechanisms cannot be directly applied to autonomous driving, as they cannot ensure end-to-end deadlines due to significant runtime execution time variation. Moreover, current adaptive approaches cannot effectively determine the optimal point for timely control command generation. In contrast, HCPerf intelligently prioritizes responsiveness and throughput by taking into account runtime driving performance and user comfort.

IX. Conclusion

In this paper, we propose HCPerf, a performance-directed hierarchical coordination framework for autonomous driving. It intelligently coordinates the autonomous driving tasks with

high execution time variation and complex dependencies according to the driving performance. Specifically, the internal coordinator intelligently schedules the tasks according to the driving performance. The external coordinator dynamically adjusts the rates of tasks according to the schedulability of the system. The experiment results show that HCPerf effectively improves the driving performance by 7.69%-45.94%.

ACKNOWLEDGMENT

This paper is supported the Science and Technology Development Fund of Macau SAR (File no. 0021/2022/ITP, 0081/2022/A2, SKL-IOTSC(UM)-2021-2023, 0123/2022/AFJ, and 0015/2019/AKP), GuangDong Basic and Applied Basic Research Foundation (No. 2020B515130004), and Key-Area Research and Development Program of Guangdong Province (No. 2020B010164003), and SRG2022-00010-IOTSC. Please ask Dr. ChengZhong Xu (czxu@um.edu.mo) and Dr. Li Li (llili@um.edu.mo) for correspondence.

REFERENCES

[1] Apollo, "an open autonomous driving platform source-code and manuals, 2022, [online]," [EB/OL], https://github.com/ApolloAuto/apollo.

[2] M. Alcon, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Timing of autonomous driving software: Problem analysis and prospects for future solutions," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 267–280.

[3] A. König, L. Nicoletti, D. Schröder, S. Wolff, A. Waclaw, and M. Lienkamp, "An overview of parameter and cost for battery electric vehicles," *World Electric Vehicle Journal*, vol. 12, no. 1, p. 21, 2021.

[4] Y. Xing, C. Huang, and C. Lv, "Driver-automation collaboration for automated vehicles: a review of human-centered shared control," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 1964–1971.

[5] K. N. de Winkel, T. Irmak, R. Happee, and B. Shyrokau, "Standards for passenger comfort in automated vehicles: Acceleration and jerk," *Applied Ergonomics*, vol. 106, p. 103881, 2023.

[6] Q. Xia, D. Zhu, and H. Aydin, "Work-in-progress: Preference-oriented scheduling in multiprocessor real-time systems," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 135–138.

[7] H. Choi, H. Kim, and Q. Zhu, "Job-class-level fixed priority scheduling of weakly-hard real-time systems," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 241–253.

[8] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 35–46.

[9] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 67–79.

[10] A. V. Malawade, T. Mortlock, and M. A. A. Faruque, "Ecofusion: Energy-aware adaptive sensor fusion for efficient autonomous vehicle perception," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 481–486.

[11] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007, pp. 289–302.

[12] M. Chen, C. Nolan, X. Wang, S. Adhikari, F. Li, and H. Qi, "Hierarchical utilization control for real-time and resilient power grid," in *2009 21st Euromicro Conference on Real-Time Systems*. IEEE, 2009, pp. 66–75.

[13] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of dags on clustered multi-core platforms," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 156–168.

[14] H. Tan, G. Lu, and M. Liu, "Risk field model of driving and its application in modeling car-following behavior," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 605–11 620, 2021.

[15] G. Utrera, M. Farreras, and J. Fornes, "Task packing: Efficient task scheduling in unbalanced parallel programs to maximize cpu utilization," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 37–49, 2019.

[16] K. Huang and Q. Hao, "Joint multi-object detection and tracking with camera-lidar fusion for autonomous driving," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6983–6989.

[17] M. Fliess and C. Join, "Model-free control," *International Journal of Control*, vol. 86, no. 12, pp. 2228–2252, 2013.

[18] J. Wang, H. Mounier, S.-I. Niculescu, M.-S. Geamanu, and A. Cela, "Event-driven model-free control in motion control with comparisons," *IMA Journal of Mathematical Control and Information*, vol. 34, no. 4, pp. 1255–1275, 2017.

[19] M. Fliess, C. Join, and H. Sira-Ramirez, "Non-linear estimation is easy," *International Journal of Modelling, Identification and Control*, vol. 4, no. 1, pp. 12–27, 2008.

[20] Z. Wang and J. Wang, "Ultra-local model predictive control: A model-free approach and its application on automated vehicle trajectory tracking," *Control Engineering Practice*, vol. 101, p. 104482, 2020.

[21] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[22] Firefly, "Core, 2022, [online]," [EB/OL], https://www.t-firefly.com.

[23] Nvidia, "Nvidia jetson tx2, 2022, [online]," [EB/OL], https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-tx2/.

[24] L. Krawczyk, M. Bazzal, R. P. Govindarajan, and C. Wolff, "An analytical approach for calculating end-to-end response times in autonomous driving applications," in *10th International Workshop on Analysis Tools and Methodologies for Embedded and Realtime Systems (WATERS 2019)*, 2019.

[25] S. Gawiejnowicz and S. Gawiejnowicz, "The modern scheduling theory," *Models and Algorithms of Time-Dependent Scheduling*, pp. 65–82, 2020.

[26] F. Wurst, D. Dasari, A. Hamann, D. Ziegenbein, I. Sanudo, N. Capodieci, M. Bertogna, and P. Burgio, "System performance modelling of heterogeneous hw platforms: An automated driving case study," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 365–372.

[27] R. Utriainen, M. Pöllänen, and H. Liimatainen, "The safety potential of lane keeping assistance and possible actions to improve the potential," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 4, pp. 556–564, 2020.

[28] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 107–118.

[29] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu, "Co-optimizing performance and memory footprint via integrated cpu/gpu memory management, an implementation on autonomous driving platform," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 310–323.