Driving Scenario Perception-Aware Computing System Design in Autonomous Vehicles

Hengyu Zhao, Yubo Zhang†, Pingfan Meng†, Hui Shi, Li Erran Li†, Tiancheng Lou†, Jishen Zhao
University of California, San Diego †Pony.ai
{*h6zhao, hshi, jzhao*}*@ucsd.edu*
†{*yubo, pmeng, erranlli, acrush*}*@pony.ai*

***Abstract*—Recently, autonomous driving ignited competitions among car makers and technical corporations. Low-level autonomous vehicles are already commercially available. However, high autonomous vehicles where the vehicle drives by itself without human monitoring is still at infancy. Such autonomous vehicles (AVs) fully rely on the computing system in the car to perceive the environment and make driving decisions. In AV computing systems, the latency is an essential metric for ensuring the efficiency and safety, because a timely decision with low latency will avoid accidents and save lives. Moreover, we perform a field study by running industrial Level-4 autonomous driving fleets in various locations, road conditions, and traffic patterns. We observe that the perception module consumes the longest latency, and it is highly sensitive to surrounding obstacles. To study the correlation between perception latency and surrounding obstacles, we propose a perception latency model. Moreover, we demonstrate the use of our latency model, by developing and evaluating a driving scenario perception-aware AV computing system that efficiently manages computation hardware resource. Our evaluation results show that the proposed AV system resource management improves performance significantly.**

I. Introduction

We are on the cusp of a transportation revolution where the autonomous vehicle (also called self-driving car, uncrewed car, or driverless car) is likely to become an essential mobility option [29]. Autonomous vehicles (AVs) allow people to sit back and tell their cars where to drive themselves. AVs incorporate sophisticated suites of sensors, such as cameras, light detecting and ranging (LiDAR), and radar [41] (Figure 1); These sensors are backed by advanced computing system software and hardware that interpret massive streams of data in real-time. As such, autonomous driving promises new levels of efficiency and takes driver fatigue and human errors out of the safety equation.

However, the autonomous driving will shift the burden of safety guarantee towards the AV computing system. To ensure safety, the AV computing system is required to make not only appropriate driving decisions, but also real-time reactions [37]. Thus, in addition to sound AV algorithms, we still need to develop efficient computing system to reduce its latency. At least, AV computing system should react faster than human drivers. For instance, the latency should be less than 100ms, as the fastest action by a human driver usually takes longer than 100ms [29]. Therefore, this paper focuses on optimizing the computing system latency of high automation vehicles, i.e., Level-4 autonomous vehicles.

To investigate latency implications on the computing system design, we performed a field study (§ III) by running real industrial Level-4 autonomous driving fleets under testing in various locations, road conditions, and traffic patterns. We observe that perception, especially the LiDAR perception contributes most of the computing system latency (Figure 3 (a)). Moreover, we find that perception latency is highly sensitive to different driving scenarios, and each driving scenario is a specific combination of various obstacles, such as other vehicles, trees, buildings. The density and distribution of surrounding obstacles will vary in different driving scenarios.

To guide driving scenario perception aware AV computing system design, first, we present observations according to our field study. Second, we propose a latency model that analyzes the correlation between perception latency and surrounding obstacles. Finally, to fully utilize the latency model, we propose an AV system resource management scheme that is aware of surrounding obstacles. This resource management scheme includes an offline planning stage and an online monitoring and scheduling stage. In summary, this paper makes the following contributions:

- We perform a field study with our AV fleets and provide a tutorial of state-of-the-art AV systems. We present observations and challenges based on the extensive data collected from our field study, and introduce the AV computing system software and hardware organizations.
- We propose a perception latency model, which allows AV computing system architects to estimate the perception latency of a given computing system and architecture design in various driving scenarios.
- We demonstrate an example of utilizing our latency model to guide AV system computation resource management with a hardware and software co-design.

II. State-of-the-art AV System

To understand the current status of AV development, we examine the state-of-the-art AV design, including AV automation level taxonomy, AV computing system architecture and introduction to tasks of AV computing system.

A. Levels of Automation

The Society of Automotive Engineers (SAE) categorizes autonomous driving systems into six automation levels, which range from no automation (Level-0) to full automation (Level-5) [16]. At Level-2 and Level-3, the human driver still needs

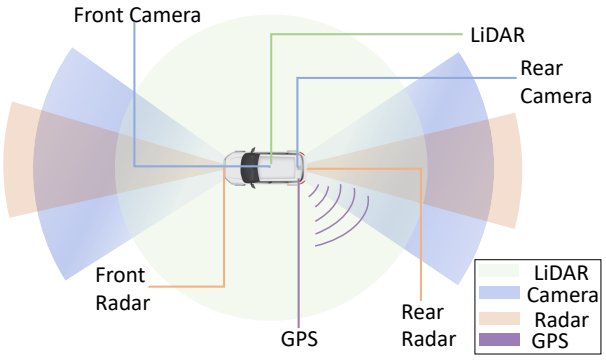


Fig. 1. Top view of a typical autonomous vehicle with various sensors.

TABLE I
COMPUTING SYSTEM AND ARCHITECTURE CONFIGURATIONS.

CPU	Intel Xeon processor
Main memory	16GB DDR4
Operating system	Ubuntu
GPU	NVIDIA Titan V (Volta architecture)
Device memory	12GB HBM2

to respond to emergencies. A Level-4, AV drives itself almost all the time without any human input, so it will highly rely on the computing system to perform most driving tasks. Full automation vehicles (Level-5), which can operate on any road and in any conditions that a human driver could manage, is a long-term goal of autonomous driving development. This paper focuses on Level-4 AVs, which introduce much more critical latency and safety requirements on the computing system than lower-level AVs.

B. AV Computing System Architecture

While embedded or low-end processors may be sufficient to meet the computation requirement of Level-2 and Level-3 AVs, current Level-4 AV system designs typically adopt high-end heterogeneous architectures, which comprise sophisticated CPUs, GPUs, storage devices (e.g., terabytes of SSDs), FPGAs [1] and other accelerators, to accommodate the intensive computation demand. As such, high automation AV systems are fundamentally different from conventional real-time embedded systems. Table I lists the heterogeneous architecture employed by our prototypes.

Implication: Although the computing system adopts server-grade processors, it is challenging to minimize the latency of all computation tasks due to the vast amount of interdependently executing software modules (discussed in § IV-A) and the finite hardware resources in the system.

C. Tasks of AV Computing System

The computing system interacts with various sensors and the car control system to perform three primary functions: localization, perception, and planning and control. Figure 2 illustrates the general working flow of the state-of-the-art Level-4 AV computing system.

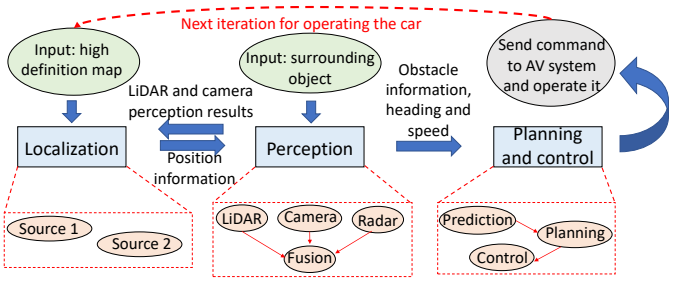


Fig. 2. AV computing system tasks.

Localization. Localization identifies the location of the vehicle on a high definition map, based on the information obtained from LiDAR, camera, and GPS. The high definition map stores static information on the way, such as lane lines, trees, guardrails, and the location of traffic lights and stop signs [8]. GPS offers a prediction of the current location of the vehicle on a global scale; but such prediction is not sufficiently accurate to allow the computing system to perform driving tasks, e.g., staying within a lane on the road [9]. Therefore, GPS information needs to be accompanied with LiDAR and camera "perception" (described below) of surrounding static objects, to interpret the local environment accurately.

Perception. Perception detects and interprets surrounding static (e.g., lane lines, trees, and traffic lights) and moving (e.g. other vehicles and pedestrians) objects with three types of sensors, LiDAR, camera, and millimeter-wave radar, as shown in Figure 1. In the rest of the paper, we refer all the objects, which can be detected by perception, i.e., in the detection range, as "obstacles". Localization typically only requires infrequent perception of static objects, while traffic interpretation requires frequent (e.g., per-frame – each frame of data collected by the sensors) perception of both static and moving objects.

Compared to camera and radar, LiDAR offers much higher accuracy on detecting the position, 3D geometry, speed, and moving direction of obstacles [21], [36]. Therefore, most Level-4 AVs highly rely on LiDAR to perform the perception task [15], [19], [23]. LiDAR continues to emit lasers across a 360 degree view. It receives reflected signals, whenever its laser is blocked by an obstacle. Based on the received signals, we can build a "point cloud" to represent the location and 3D geometry of the obstacle [35]; each point in the graph represents a received signal.

Overall, perception obtains accurate information of location, geometry, speed, and moving direction of obstacles. Such information is essential for localization and making appropriate driving decisions to avoid collisions in planning and control.

Planning and control. The essential functions of planning and control include prediction, trajectory planning, and vehicle control. Their dependency relationship is shown in Figure 2. With the perception results, the prediction function tracks the behavior of obstacles at real-time and predicts their next movements. The computing system plans an optimal trajectory for the vehicle, based on the prediction and perception results.

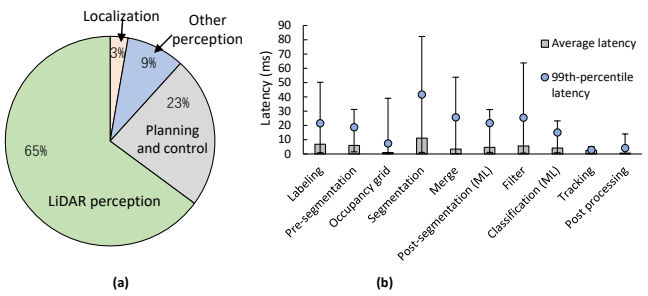


Fig. 3. (a) Computing system latency breakdown. (b) Latency of LiDAR perception main modules.

The control function will control the steering, acceleration, and deceleration of the vehicle based on the driving decision made by the planning function.

III. Field Study and Observations

To explore realistic AV latency/safety requirement and computing system hardware/software behaviors, we run a fleet of Level-4 AVs in various locations, road conditions, and traffic patterns over three contiguous months. Our field study yields over 200 hours and 2000 miles of traces with a considerable size of data. The data is collected at the granularity of one sensor frame, which is the smallest granularity that the computing system interprets and responds to obstacles. With each frame, we collect (1) the execution latency of each software module, (2) total computing system latency, (3) the utilization of computing system hardware resources (CPU, GPU, and memories), (4) the environment information obtained by the sensors (e.g., the distribution of static and moving obstacles), and (5) instantaneous velocity and acceleration of the AV.

Breakdown of computing system latency and the impact of architecture design. As shown in Figure 3(a), the majority (74%) of computing system latency is consumed by perception. LiDAR perception is the major contributor to perception latency. In addition, we observe that most of the workloads running in our AV system are compute-intensive. Various software modules have a large variety of computation requirements and parallelism characteristics. As a result, native heterogeneous system hardware resource management easily results in unbalanced compute resource utilization, leading to significant delays in software execution. Figure 3(b) shows an example with LiDAR perception modules by illustrating average, maximum and minimum, and tail latency of the ten main software modules across thousands of executions during our field study. We observe that latency-critical modules also typically have strict inter-dependency with other modules in the dependency graph. Examples include *Segmentation* and *Filter* (Figure 4). Based on the two observations, latency-critical modules can be identified as those having long maximum latency and strict inter-dependency with other modules.

Timely traffic perception is the most critical to AV safety. Due to the limited ranges of sensor detection, the computing system has limited time (typically within one or several sensor sampling intervals) to interpret an obstacle in the traffic after it enters the sensor-detectable region. Moreover, as perception consumes the majority of computing system latency, timely traffic perception is the most critical to AV safety. Specifically, LiDAR perception is the most safety critical module in our AV system (and many other Level-4 AV implementations), because most of the perception task relies on LiDAR. In fact, we encountered several safety emergencies, when the AV needed to perform a hardbrake due to the delay of LiDAR perception.

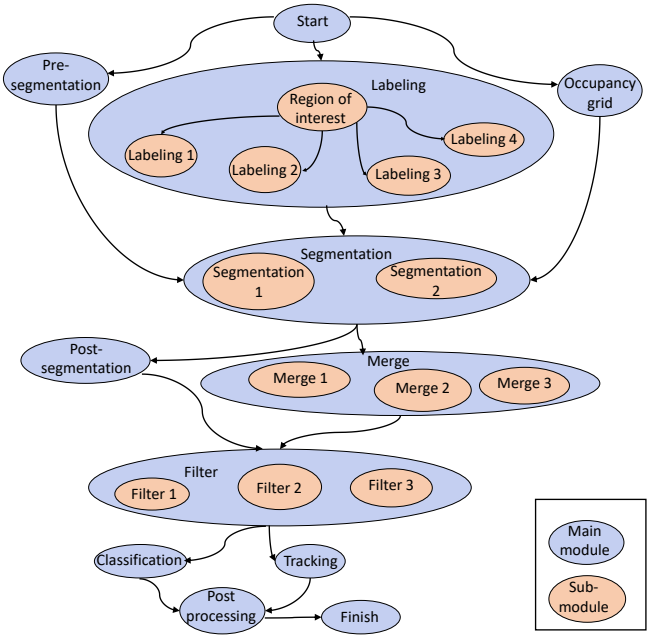


Fig. 4. Dependency graph of LiDAR perception.

Correlation between perception latency and obstacle distribution. We observe that perception latency depends on both computing system configuration and the distribution of obstacles (especially in traffic). With a given computing system hardware configuration, obstacles closer to the vehicle have a higher impact on perception latency than distant ones. Moreover, among the close obstacles, the denser the obstacle distribution, the longer the perception latency would be. The rationale underlying is that close, and dense obstacles reflect more LiDAR signals than other obstacles, hence generate more data for LiDAR perception.

IV. Latency Model

The goal of our latency model is to estimate the perception latency, given obstacle distribution and computation resource allocation, because our field study shows that perception latency is a significant contributor to system latency (§ III).

Our model is built based on two observations: first, closer and denser obstacles lead to higher perception latency (§ III); second, the perception latency also depends on computation

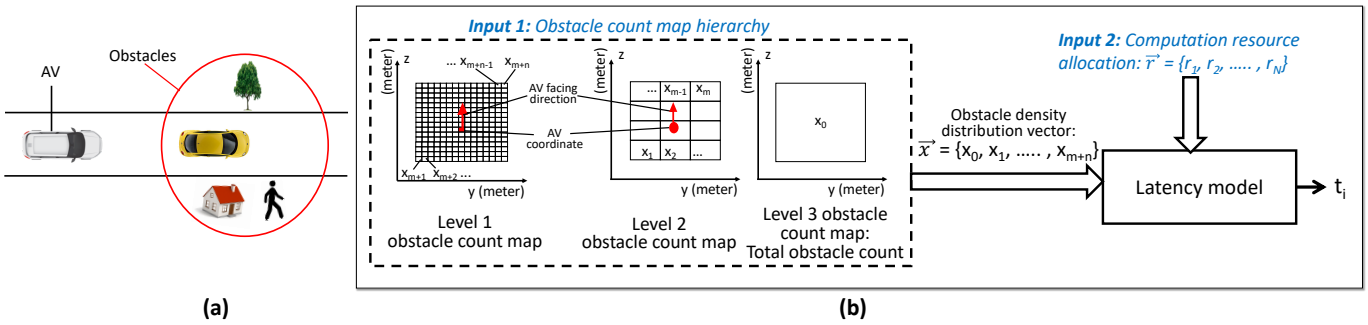


Fig. 5. Overview of the proposed perception latency model. (a) Sources of perception latency. (b) Latency model working flow.

resources allocated to run a software module[1]. Figure 5 shows an overview of our latency model. The model input includes (1) obstacle density distribution and (2) computation resource allocation of each software module. The output is the estimated latency of each software module. Our latency model estimates t_i in two steps:

- We first estimate a baseline latency (τ_i), which is the latency of executing a software module on a baseline computation resource (e.g., CPU in our AV system), using a baseline latency model presented by Equation 1.
- Then, we calculate t_i with latency conversion ratios based on given resource allocation plan.

In the following, we use LiDAR perception as an example to discuss our latency model in detail and can surely apply it to other perception modules, such as camera, radar, etc. § VI-A evaluates the accuracy of our latency model.

A. LiDAR Perception

We present how LiDAR perception works by illustrating its software modules and dependency graph.

Dependency graph. To process LiDAR data, the computing system executes a considerable number of inter-dependent software modules (i.e., software programs) at real-time. To understand the dependency relationship among these software modules, we build a dependency graph based on our AV system. Figure 4 shows an example[2]. LiDAR perception consists of a set of main modules (shown in blue) and sub-modules (shown in orange). For example, the *Labeling* module first sorts out a rectangular region-of-interest (ROI), which restricts the range of LiDAR detection; then, it calls its four sub-modules to perform finer-grained labeling within the ROI. As another example, *Segmentation* extracts useful semantic and instance information (e.g., surrounding vehicles and pedestrians) from the background. The *Merge* module clusters the point clouds, which are likely to belong to the same object; the *Filter* modules reduce noise in the point clouds. Some software modules, e.g., *Segmentation*, do not only impose long latency, but also have a strong dependency with other software modules.

Implication: Due to their long latency and high dependency, certain software modules contribute more to the overall perception latency and computing system latency than other modules. Therefore, these modules are typically more safety critical.

B. Model Input

Obstacle density distribution vector. We design an obstacle count map to represent obstacle density distribution. The map divides the sensor ROI into regular grids, with each grid storing the number of obstacles in it. For example, our AV has a $64m \times 50m$ ROI with a $2m \times 2m$ grid size. This grid size captures most moving objects on the road, e.g., vehicles and bicycles. But large obstacles, which spread across multiple grids, are counted multiple times. To express more accurate obstacle density, we adopt a hierarchy of three levels of obstacle count maps with a grid size of $2m \times 2m$, $8m \times 10m$, and $64m \times 50m$ (the whole ROI), respectively. With the map hierarchy, we generate an obstacle density distribution vector $\vec{x}$ as illustrated in Figure 5(b).

Computation resource allocation vector. We represent computation resource allocation by a resource index vector $\vec{r}$, where each element in the vector indicates the resource allocation of a software module. For example, our current AV implementation only has two resource allocation options, running on CPU (index=0) or GPU (index=1). Then, $\vec{r} = [0,0,1,...,0]$ indicates that software modules are executed on CPU, CPU, GPU, ..., CPU.

C. Latency Model

Perception software modules vary in complexity. We take into account the diversity in the algorithm complexity and model the baseline latency of each software module (τ_i) by the following equation,

$$\tau_i = \vec{a} \cdot (\vec{x} \odot \vec{x}) + \vec{b} \cdot (\vec{x} \odot log(\vec{x})) + \vec{c} \cdot \vec{x} + \vec{d} \cdot log(\vec{x}) + e \quad (1)$$

Here, x is a $m+n+1$ dimension obstacle density distribution vector, where m and n are the grid counts in the first-level and second-level obstacle count map, respectively. The operator

[1]In our current AV system, such hardware resource allocation refers to whether executing a software module on CPU or GPU. Although our system does not adopt FPGA or other accelerators, our latency model applies to general heterogeneous architecture design with a variety of hardware resources.

[2]We only show renamed high-level modules due to confidential reasons.

$\odot$ is coefficient-wise vector product, and $\cdot$ is the inner vector product.

This is a standard curve fitting problem. To solve for coefficients $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, and e in the equation, we perform linear regression on massive road testing data, which is used to generate the model input data, i.e., obstacle density distribution and corresponding baseline latency.

The final latency of each perception module then can be estimated by

$$t_i = \tau_i \nu(r_i) \tag{2}$$

Here, $\nu(r_i)$ is the latency conversion ratio of executing the same module on a different computation resource than the baseline. This ratio is determined by exhaustive search of running a software module with various computation resource allocation configurations.

V. AV System Resource Management

We demonstrate an example of utilizing our latency model to guide the computing system design, by designing a heterogeneous computation resource management scheme. The goal of resource management is to determine the computation hardware resource allocation of each software module and the priority of software module execution to optimize perception latency, as well as the computing system latency.

Overview. Our resource management has two phases, resource planning and resource scheduling. During the *planning phase*, we perform an exhaustive search of computation resource allocation and priority options of executing software modules in various obstacle distributions, to determine the options that minimize perception latency given each obstacle distribution. This yields a resource management plan for each obstacle distribution. During the *scheduling phase*, we match the current obstacle distribution with resource management plans and determine the plan to use. To ensure sufficient resources to be scheduled for each software module, our AV system maintains redundant computation resources. Due to the large searching space, resource planning usually takes a long time. Therefore, we perform the planning phase offline, while the AV system only performs online scheduling. To further accelerate the online scheduling, offline planning classifies obstacle distributions associated with the same resource management plan to clusters. As such, the online scheduling phase only needs to match clusters.

A. Offline Planning

The offline planning phase analyzes the massive data collected by road tests to create a set of resource management plans with various obstacle distributions. A resource management plan includes (i) the computation resources allocated for software module execution (CPU or GPU in our current AV system design) and (ii) the priority of software modules, which is used to determine which module gets executed first, when multiple choices exist.

Algorithm 1 illustrates the offline planning procedure. For each field study sample, which comprises instantaneous computing system response time and the latency of each software module for an obstacle distribution, we perform an exhaustive search in the space of resource management plans. If several samples achieve the lowest perception latency with the same resource management plan, we categorize these samples in the same cluster. Based on our field study, samples in the same cluster have similar (1) obstacle density distribution and (2) perception timeout pattern, which is the number of continuous incidents where the perception latency exceeds the sensor sampling interval (e.g., a LiDAR sampling interval is 100ms). Therefore, we concatenate these two as the feature, and compute the feature vector of each cluster by averaging the feature of each sample in the cluster.

This phase only need to be done once, whenever the AV adapts to a new location or physical system. In practice, we found that the clustering and the associated resource management plans are stable over time, even though users can also periodically run the offline phase to re-calibrate.

Algorithm 1 Offline planning.

Input : *Obstacle density distribution vector* $S \in R^{N\times m}$
Input : *Timeout pattern* $K \in R^{N\times m}$
Input : *Dependency graph* G
Input : *Computation resource management plans* P
Input : *Perception latency function* L
Output : *Category features* $F \in R^{k\times 2m}$
Init : *Category Dictionary* D
for $i = 1 \rightarrow N$ **do**
    $P_i^* = \text{argmax}_{P_j} L(G, P_j, S_i)$
    append $S_i \oplus K_i$ *to* $D[P_i^*]$
for $j = 1 \rightarrow k$ **do**
    for $q = 1 \rightarrow m$ **do**
        $F_{j,q} \leftarrow mean(D[P_j]_q)$

B. Online Monitoring and Scheduling

The online phase adopts a software/hardware co-design approach to (1) monitor the computing system execution and (2) match the resource management plans. Figure 6 shows a system design overview. When the AV is driving, the computing system continuously (1) monitors instantaneous system latency and the pattern of perception timeout and (ii) determines when and which resource management plan to adopt.

A loop of three steps implements this: (Step-1) perception timeout pattern monitoring, (Step-2) cooperative cluster matching, and (Step-3) plan switching. The online phase triggers a hardware-based perception timeout pattern monitoring, whenever one of the safety critical perception software modules has a timeout (e.g. LiDAR); the monitoring hardware counts the number of continuous timeouts across the subsequent LiDAR sampling intervals to identify whether timeout happens accidentally or continuously, e.g., over a user-defined h continuous timeouts. Based on our field study, LiDAR perception timeout

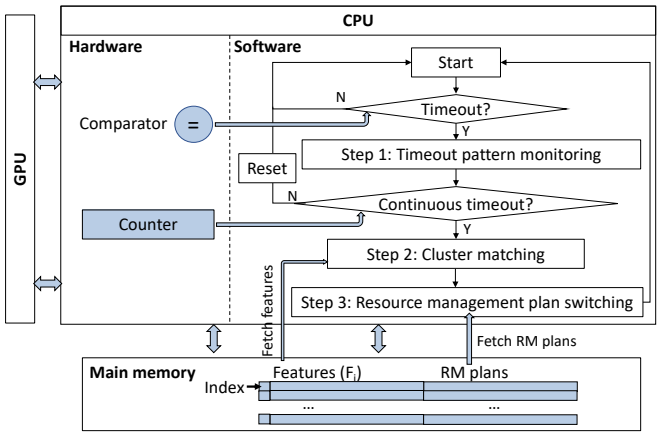


Fig. 6. AV computing system design that performs online monitoring and scheduling.

happens either accidentally for a few times or continuously for more than 100 frames. Therefore, we set h to be 100. If continuous timeout happens, the online phase performs the next two steps based on the following equations:

$$index = \underset{i}{\operatorname{argmin}} \|F_{current} - F_i\| \tag{3}$$

$$p_i = D[index] \tag{4}$$

where $F_{current} = S_{current} \oplus K_{current}$. This is the current feature vector, which is the concatenation of current obstacle density distribution vector $S_{current}$ and current timeout pattern $K_{current}$; $\oplus$ is the concatenation operator. F_i is the feature vector of the i_{th} cluster, which consists of the obstacle density distribution vector and timeout pattern of the i_{th} cluster. The online resource management software will scan the features of all clusters and identify a matching cluster, by finding the cluster with the closest feature as the current one. We assume road condition and traffic pattern do not dramatically change within a certain amount of time, so the plan switching is infrequent.

Hardware and software implementation. To meet the real-time requirement of online resource management, we implement Step-1 in CPU hardware by adding a set of comparators and counters as shown in Figure 6. The comparators track the timeouts; the counters accumulate the number of continuous timeouts. In our AV system, an 8-bit counter is sufficient to capture the continuous timeout patterns. Step-2 and Step-3 are triggered once Step-1 detects continuous timeouts. As a result, these Step-2 and Step-3 are performed infrequently. As such, we implement the primary functionality of these two steps by software as a load balancer [28], [31] in CPU; it acts as a reverse proxy of software module scheduling across CPU and GPU. The user-level implementation imposes negligible performance overhead demonstrated by our experiment results, due to the infrequent cluster matching and resource management plan switching. In our AV system, the software modules that can be executed on both CPU and GPU are implemented and compiled with both versions. We store the indices generated during Step-2 in CPU main memory. Step-3 reads the memory to identify the resource management plan with the current feature.

VI. Evaluation

We evaluate our design by two simulators: an AV simulator and an architecture simulator. Our AV simulator investigates whether there exists a collision with various AV configurations/parameters and road conditions. It is developed by the company, similar to the Virtual Reality Autonomous Vehicle Simulator built by NVIDIA [2]. Our architecture simulator evaluates the effectiveness of various resource management schemes (RMs). It is a trace-based simulator; each trace includes the priority of each software module execution, as well as its execution latency, data access latency, RM switching latency on either CPU or GPU, and energy consumption obtained by NVprof and Vtune in our native AV system. Our simulator models our AV CPU+GPU system architecture and our RM hardware design, including the latency of each access to the hardware counters, comparators, tables in the memory. We compare three computing system architecture configurations and resource management policies:

- **CPU** – Executing all software modules on CPU;
- **CPU+GPU** – The native heterogeneous AV system design. It employs tail latency to determine the priority of software module execution in resource management, with the GPU to accelerate all deep learning software modules.
- **Resource management** – Heterogeneous AV computing system design with our resource management scheme. The results take into account the performance overhead of the online monitoring and scheduling phase.

A. Latency Model Analysis

Pearson correlation coefficient. We adopt the Pearson correlation coefficient [7] to evaluate the correlation between LiDAR perception latency and obstacle density distribution. Figure 7 depicts a heat map of Pearson correlation coefficients distributed in an obstacle count map hierarchy, based on real driving scenarios during our field study. Instead of the count of obstacles, each grid in Figure 7 stores a Pearson correlation coefficient. The coefficient measures the correlation between obstacle density and LiDAR perception latency: the higher the coefficient, the larger the correlation between the two. Figure 7 uses colors to express Pearson correlation coefficient values: the darker the color, the larger the value. As such, in the areas with darker colors, LiDAR perception latency is more susceptible to obstacle density. We make three observations from Figure 7. First, LiDAR perception latency is more susceptible to nearby obstacles than those far away. This is in line with our field study observation. Second, both the top left and top right areas have high Pearson coefficient. We believe this is caused by heavy horizontal traffic through an intersection during rush hours. Finally, LiDAR perception latency is more sensitive to the density than the distribution of obstacles.

Model accuracy. We use mean squared error (MSE) [39] (the lower the better) to evaluate the accuracy of our perception latency model quantitatively. We investigate our latency model

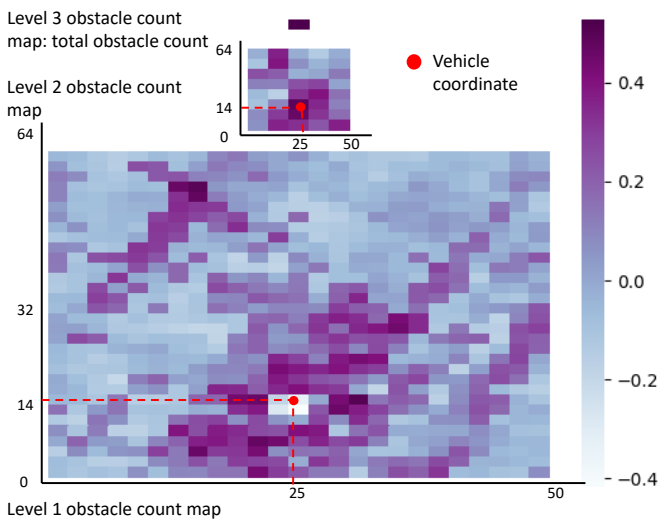


Fig. 7. Heat map of pearson correlation coefficient for obstacle count map hierarchy.

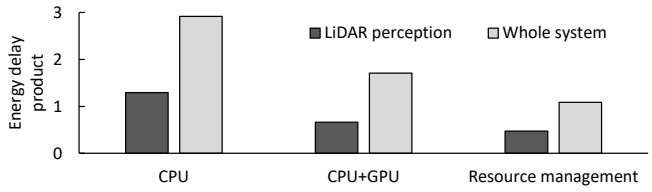


Fig. 8. Energy efficiency of various AV computing system designs.

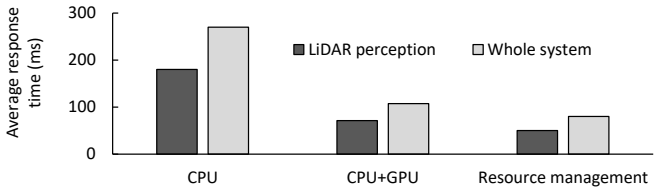


Fig. 9. Average latency of various computing system designs.

under a variety of driving scenarios, such as morning and afternoon, rush hours and non-rush hours, and local roads and express ways. The proposed latency model consistently provides high accuracy with an average MSE as low as 1.7×10^{-4}.

B. Resource Management Analysis

Beyond improving safety, *Resource Management* also improves computing system performance and energy efficiency. Figure 8 compares the energy-delay product (EDP) of various configurations. *Resource Management* leads to the lowest EDP in both LiDAR perception alone and the whole computing system. Our hardware modifications to implement online resource management impose less than 1% of total system energy consumption. Figure 9 compares the average latency of various configurations. Compared with *CPU* and GPU+CPU, *Resource management* reduces the latency averaged throughout our field study data by 2.4× and 35%, respectively; LiDAR perception latency is reduced by 2.6× and 45% on average.

VII. Related Work

To our knowledge, this is the first paper to identify the correlation between obstacles and computing latency, and propose a latency model used for guiding AV computing systems and architecture design, based on a large-scale field study of real industry Level-4 AVs. In this section, we discuss related works on AV system design, AV computing system design and heterogeneous computing system design.

AV system design. Most previous AV system designs focus on developing software module algorithms [3], [12], [17], [18], [20], [26], [27], [32]. For instance, recent works improve the latency and precision of LiDAR- and camera-based perception, by leveraging video sensing, laser rangefinders, light-stripe range-finder to monitor all around the autonomous driving vehicle to improve safety with a map-based fusion system [4], [26], [32]. Several studies enhance the quality of driving decisions by optimizing perception and planning software algorithms [17], [18]. Previous works also investigate ways of effectively responding to uncertainty at real-time [3], [20], [27] and path planning to avoid static obstacles [12].

AV computing system design. Most previous studies on AV computing systems and architecture design focus on accelerator design, and energy efficiency improvement of deep learning or computer vision software execution [22], [29], [33], [40]. These studies accelerate some general AV algorithms, such as iterative closet point (ICP) algorithm [11] and YOLO object detection algorithm [34], by designing domain specific accelerators. However, these studies do not consider how the density and distribution of surrounding obstacles affect the perception latency. Our latency model can also be used to guide deep learning and computer vision accelerator design in AV systems.

Heterogeneous computing system design. AV computing systems adopt sophisticated heterogeneous architectures used in real-time scenarios. Prior works on real-time heterogeneous computing systems [10], [14], [38] focus on embedded processors, rather than sophisticated architectures. Previous studies that investigate server-grade CPU and GPU processor based systems focus on distributed and data center applications [5], [6], [13], [24], [25], [30]; in such cases, scalability is a more critical issue than single-system resource limitation. Furthermore, neither type of heterogeneous system use cases has the level of safety requirements as autonomous driving.

VIII. Conclusion

In summary, we present the state-of-the-art AV system, as well as discuss a set of implications of AV system design based on a field study with industrial AVs. Furthermore, we propose a perception latency model to guide AV computing system design. Finally, we also demonstrate the use of our latency model with a heterogeneous computation resource management scheme. As high automation vehicles are still at the early stages

of development, we believe our work only scratches the surface of AV computing system design. Substantial follow-up work is required to enhance the computing system design towards safe and efficient autonomous driving.

IX. Acknowledgement

We thank the anonymous reviewers for their valuable feedback. This paper is supported in part by NSF grants 1829524 and 1829525.

References

[1] "Alphabet's Waymo is now in the 'prove it' stage of its self-driving journey," 2018. [Online]. Available: https://www.thestreet.com/opinion/alphabet-s-waymo-is-now-in-the-prove-it-stage-of-its-self-driving\\-journey-14736954

[2] "Virtual reality autonomous vehicle simulator," 2019. [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/

[3] M. Althoff, O. Stursberg, and M. Buss, "Model-based probabilistic collision detection in autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, pp. 299–310, 2009.

[4] R. Aufrère, J. Gowdy, C. Mertz, C. Thorpe, C.-C. Wang, and T. Yata, "Perception for collision avoidance and autonomous driving," *Mechatronics*, vol. 13, no. 10, pp. 1149–1161, 2003.

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[6] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*. IEEE Computer Society, 2010, pp. 826–831.

[7] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[8] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017.

[9] W. Burgard, O. Brock, and C. Stachniss, "Map-based precision vehicle localization in urban environments," in *MIT Press*, 2011, pp. 352–359.

[10] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *Journal of Systems and Software*, vol. 99, pp. 20–35, 2015.

[11] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," in *Object recognition supported by user interaction for service robots*, vol. 3. IEEE, 2002, pp. 545–548.

[12] K. Chu, M. Lee, and M. Sunwoo, "Local path planning for off-road autonomous driving with avoidance of static obstacles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1599–1616, 2012.

[13] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1998, pp. 62–82.

[14] Y. Dang, B. Wang, R. Brant, Z. Zhang, M. Alqallaf, and Z. Wu, "Anomaly detection for data streams in large-scale distributed heterogeneous computing environments," in *ICMLG*, 2017, p. 121.

[15] L. Delobel, C. Aynaud, R. Aufrere, C. Debain, R. Chapuis, T. Chateau, and C. Bernay-Angeletti, "Robust localization using a top-down approach with several lidar sensors," in *ROBIO*. IEEE, 2015, pp. 2371–2376.

[16] A. Driving, "Levels of driving automation are defined in new SAE international standard J3016: 2014," *SAE International. Google Scholar*.

[17] A. Furda and L. Vlacic, "Towards increased road safety: Real-time decision making for driverless city vehicles," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 2009, pp. 2421–2426.

[18] A. Furda and L. Vlacic, "Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making," *IEEE Intelligent Transportation Systems Magazine*, vol. 3, no. 1, pp. 4–17, 2011.

[19] J. Hecht, "Lidar for self-driving cars," *Optics and Photonics News*, vol. 29, no. 1, pp. 26–33, 2018.

[20] T. Helldin, G. Falkman, M. Riveiro, and S. Davidsson, "Presenting system uncertainty in automotive uis for supporting trust calibration in autonomous driving," in *AutomotiveUI*. ACM, 2013, pp. 210–217.

[21] M. Himmelsbach, A. Mueller, T. Lüttel, and H.-J. Wünsche, "Lidar-based 3D object perception," in *Proceedings of 1st international workshop on cognition for technical systems*, vol. 1, 2008.

[22] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car part II: A case study on the implementation of an autonomous driving system based on distributed architecture," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5119–5132, 2015.

[23] A. Joshi and M. R. James, "Generation of accurate lane-level maps from coarse prior maps and lidar," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 19–29, 2015.

[24] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous cloud storage configuration for data analytics," in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, 2018, pp. 759–773.

[25] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2002.

[26] F. Kunz, D. Nuss, J. Wiest, H. Deusch, S. Reuter, F. Gritschneder, A. Scheel, M. Stübler, M. Bach, P. Hatzelmann *et al.*, "Autonomous driving at Ulm university: A modular, robust, and sensor-independent fusion approach," in *IV*. IEEE, 2015, pp. 666–673.

[27] C. Laugier, I. Paromtchik, M. Perrollaz, Y. Mao, J.-D. Yoder, C. Tay, K. Mekhnacha, and A. Nègre, "Probabilistic analysis of dynamic scenes and collision risk assessment to improve driving safety," *Its Journal*, vol. 3, no. 4, pp. 4–19, 2011.

[28] J. Lee, M. Samadi, Y. Park, and S. Mahlke, "Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems," in *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 245–256.

[29] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *ASPLOS*. ACM, 2018, pp. 751–766.

[30] M. Maheswaran, S. Ali, H. Siegal, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*. IEEE, 1999, pp. 30–44.

[31] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 69:1–69:35, Jul. 2015.

[32] U. Ozguner, C. Stiller, and K. Redmill, "Systems for safety and autonomous behavior in cars: The darpa grand challenge experience," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 397–412, 2007.

[33] R. Pinkham, S. Zeng, and Z. Zhang, "Quicknn: Memory and performance optimization of k-d tree based nearest neighbor search for 3d point clouds," in *HPCA*, 2020.

[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788.

[35] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[36] S. Schneider, M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Fusing vision and lidar-synchronization, correction and occlusion reasoning," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 388–393.

[37] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[38] J. Song, J. Wang, L. Zhao, S. Huang, and G. Dissanayake, "MIS-SLAM: Real-time large scale dense deformable slam system in minimal invasive surgery based on heterogeneous computing," *arXiv preprint arXiv:1803.02009*, 2018.

[39] M. Tuchler, A. C. Singer, and R. Koetter, "Minimum mean squared error equalization using a priori information," *IEEE Transactions on Signal processing*, vol. 50, no. 3, pp. 673–683, 2002.

[40] T. Xu, B. Tian, and Y. Zhu, "Tigris: Architecture and algorithms for 3d perception in point clouds," in *Micro*, 2019, pp. 629–642.

[41] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.