HydraSpace: Computational Data Storage for Autonomous Vehicles

Ruijun Wang, Liangkai Liu and Weisong Shi
Wayne State University
{ruijun, liangkai, weisong}@wayne.edu

***Abstract*— To ensure the safety and reliability of an autonomous driving system, multiple sensors have been installed in various positions around the vehicle to eliminate any blind point which could bring potential risks. Although the sensor data is quite useful for localization and perception, the high volume of these data becomes a burden for on-board computing systems. More importantly, the situation will worsen with the demand for increased precision and reduced response time of self-driving applications. Therefore, how to manage this massive amount of sensed data has become a big challenge. The existing vehicle data logging system cannot handle sensor data because both the data type and the amount far exceed its processing capability. In this paper, we propose a computational storage system called *HydraSpace* with multi-layered storage architecture and practical compression algorithms to manage the sensor pipe data, and we discuss five open questions related to the challenge of storage design for autonomous vehicles. According to the experimental results, the total reduction of storage space is achieved by 88.6% while maintaining the comparable performance of the self-driving applications.**

I. Introduction

The data captured by an autonomous vehicle is growing quickly, typically generating between 20TB and 40TB per day, per vehicle [1]. This includes cameras, which tend to generate 20 to 60Mbps, depending on the quality of the image, as well as sonar (10 to100kbps), radar (10kbps), LiDAR (10 to 70Mbps), and GPS (50kbps) [2], [3]. Storing data securely and efficiently can accelerate overall system performance. Take object detection as an example; the variation of historical data could contribute to the improvement of detection precision using machine learning algorithms. Map generation can also benefit from the stored data in updating traffic and road conditions appropriately. In addition, the sensor data can be utilized for ensuring public security as well as predicting and preventing crime. The biggest challenge is to ensure that sensors are collecting the right data, and it is processed immediately, stored securely, and transferred to other technologies in the chain, such as infrastructure, Road-Side Unit (RSU), and cloud data center. More importantly, how to create hierarchical storage and workflow that enables smooth data accessing and computing is still an open question for the future development of autonomous vehicles.

The current vehicle data logging system is designed for capturing a wide range of signals of traditional CAN bus data, including temperature, brakes, throttle settings, engine, speed, *etc.* [4]. However, it cannot handle sensor data because both the data type and the amount far exceed its processing capability. Thus, it is urgent to propose efficient data computing and storage methods for both CAN bus and sensed data to assist the development of self-driving techniques. As it refers to the installed sensors, camera and LiDAR are the most commonly used because the camera can show a realistic view of the surrounding environment while LiDAR is able to measure distances with laser lights quickly [5], [6]. Both produce a massive amount of data that would be multiplied with the increased resolution and number of channels.

Many researchers have investigated image compression to save storage spaces [7]–[11]. The authors in [12], [13] proposed an efficient image compression algorithm for gray-scale images based on the quadtree decomposition method. Other researchers have focused on bit-error aware lossless compression algorithms for color image compression subject to the bit-error rate during transmission [14]. In order to largely reduce the compressed size, lossy compression has been proposed; it does not restore the original data entirely, but the information loss has little impact on the understanding of the original image, resulting in a much larger compression ratio [15]–[17]. However, none of these methods have been utilized on the vehicular dataset to reduce the size and save storage space. With more data produced by connected and autonomous vehicles, there is a need to create an effective data store and management plan to facilitate the development of autonomous vehicles and their applications.

In this paper, we propose a novel computational data storage solution for autonomous vehicles by adopting effective compression algorithms based on different incoming sources. Comprehensive and intensive experiments were conducted to verify the effectiveness of our proposed method. The major contributions of this paper are as follows:

- Propose a computational storage architecture named

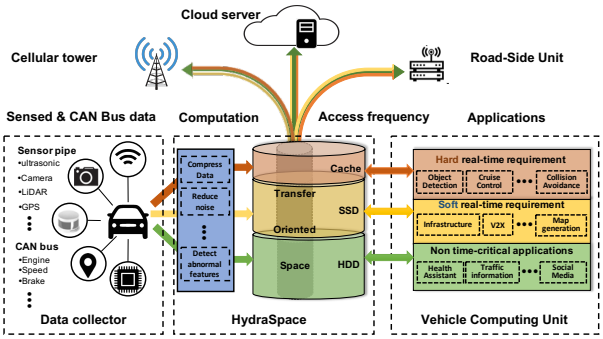


Fig. 1. An overview of *HydraSpace*.

HydraSpace to efficiently support a variety of applications leveraging diverse data sources for autonomous vehicles.

- Apply various compression algorithms to investigate the compression performance to find an optimal solution for *HydraSpace*.
- Conduct intensive experiments on an indoor Mobile platform HydraOne [18] to collect a real vehicular dataset, and test the system performance as well as the power consumption of multiple sensors.
- Discuss five open questions to envision the future storage design challenge for autonomous vehicles.

The rest of the paper is organized in the following structure. Section II focuses on the system design and implementation of our proposed data storage *HydraSpace*. The experimental setup and observation results are presented in Section III. Section IV presents the open questions and challenge regarding storage system design, Section V describes the related works and our work is concluded in Section VI.

II. HydraSpace Design

In this section, we present the system design of our proposed *HydraSpace*. It is a multilevel computational storage system that is designed for autonomous vehicles to compute and manage vehicular data based on access frequency, data type and volume, as well as real time application requirements. More importantly, how to support applying multiple machine learning models to the same data set concurrently also poses a challenge in the design of *HydraSpace*. This calls for creating an intermediate results layer in storage to avoid redundant computations. Figure 1 shows the overall architecture of *HydraSpace*.

A. Real-time Application Requirement

The storage architecture of *HydraSpace* includes a cache, solid-state drive (SSD), and traditional hard disk drives (HDD). This multilevel storage scheme is designed to cater to the vast amount of data generated by multiple sensors, various data access frequency, information backup, as well as data retrieving and analysis. As shown in Figure 1, there are three types of onboard computing applications that need the support of sensed data. To satisfy the requirements of hard real-time processing applications, such as adaptive cruise control and object detection, the data will be placed directly into a high-speed cache to accelerate the response time. For less time-critical applications that are defined as a soft real-time requirement in Figure 1, the sensor data will adopt a fast lossy compression algorithm and be stored in lower latency SSD to save storage space and meet the quick response requirement. For those non time-critical applications, the data will first be compressed using a lossless algorithm and stored in HDD to satisfy the needs of large capacity while reducing the total cost.

B. Access Frequency

The sensed data should be placed in different levels of storage architecture based on its access frequencies. The highly frequently utilized data can be arranged into a high-speed cache with limited space; others could be stored in SSDs or HDDs based on their volume and the application demands. For those hard real-time applications, the tasks will be running periodically to generate the results for the vehicle control system to take corresponding action. In other words, these data are considered as "hot" data, which will be accessed frequently by time-critical applications, such as object detection, collision avoidance, adaptive cruise control, *etc.* As we mentioned in the II-A, the less utilized data can be put into SSD or HDD for backup and later analysis.

C. Data Amount and Type

There are two major types of data flowing into *HydraSpace* in autonomous vehicles, which are sensor pipe data and CAN bus data. The sensor pipe data contains the camera, LiDAR, millimeter radar, ultrasonic, and GPS. The CAN bus data consists of traditional vehicle data such as speed, engine information, brake, temperature, *etc.* Compared to the can bus data, sensor pipe data will generate more data due to its complicated data structure and number of different sensors. In this paper, the data produced by the camera and LiDAR are the major concerns we considered in our design. The detailed analysis of the vehicular data-set structure is presented below to demonstrate the in-depth analysis of our proposed solution.

1) Camera Image dataset: The image data that we used in our paper are RGB images, which are collected using the camera installed on our mobile platform. The RGB image is a three-dimensional array: two of the dimensions specify the location of a pixel within an

image, and the other dimension specifies the color of each pixel. The color dimension always has a size of 3 and is composed of the red, green, and blue color channels of the image. We formulate the RGB image using the following equations:

$$\begin{cases} X = var_R * 0.4124 + var_G * 0.3576 + var_B * 0.1805 \\ Y = var_R * 0.2126 + var_G * 0.7152 + var_B * 0.0722 \\ Z = var_R * 0.0193 + var_G * 0.1192 + var_B * 0.9505 \end{cases} \tag{1}$$

Where X, Y stands for the pixel position within an image, Z represents the specific color of the corresponding pixel, and the parameters represent the specific color map of our selected RGB images.

2) LiDAR Point Cloud dataset: LiDAR is one of the most popular sensors installed on connected and autonomous vehicles to collect data and support the self-driving applications, such as 3D map generation, and collision avoidance. It will generate a set of points embedded in the 3D space and carry both geometry and attribute information. First, LiDAR acts as an eye of self-driving vehicles. It is continuously rotating and sending thousands of laser pulses to collide with the surrounding objects. An on-board computing system records the resulting light reflections and translates this rapidly updating point cloud into an animated 3D representation [6]. Secondly, LiDAR can be used for state localization. It can obtain precise and real-time locating information in the global coordinate system by matching the online frame with the prior map, which is needed in many modules, such as behavior decision, motion planning, and feedback control. With the help of LiDAR, autonomous vehicles travel smoothly and avoid collisions by detecting obstructions in advance. This improves the safety of commuters and makes autonomous vehicles less prone to accidents due to the risk of human negligence and reckless driving. The downside of LiDAR is that it will produce a huge amount of data up to 30MB per second (30MB/s) based on its specific configurations. In this case, managing this data in a fast and effective way could largely contribute to the development of connected and autonomous vehicles. In our proposed *HydraSpace*, we present a sensor pipe data management scheme to abide by the following rules. For hard real-time processing applications, the sensor data can be quickly accessed directly for computing purposes. For applications with soft real-time or fast processing needs, the data can be first stored into the *HydraSpace* and then retrieved by the applications when needed. The point cloud data in 3D space can be translated using the following equation:

$$(X, Y, Z) = H^{-1} * (r, \alpha, \epsilon) \tag{2}$$

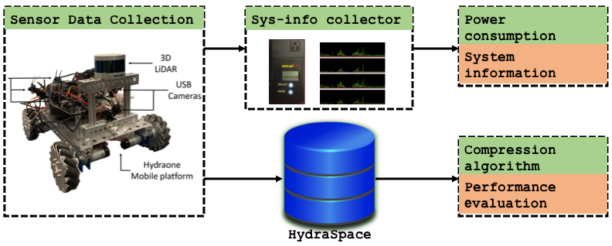


Fig. 2. The experiment setup for *HydraSpace*.

X, Y, Z stands for the position of each point in 3D space. The object can be described in the following equation:

$$O_s = [{O_s}^{(1)}, {O_s}^{(2)}, ..., {O_s}^{(N)}] \tag{3}$$

D. System Design

Based on the above requirements, we propose a computational storage *HydraSpace* with a multi-layered storage architecture and adopt effective compression algorithms to manage the sensor pipe data. The term "computational storage" means raw data will be stored after applying certain computation processes, such as compression, noise filtering, and abnormal detection. Compared with the previous method of storing raw data, the processed data can effectively improve application performance. Take data compression as an example; the compressed data can largely improve the running efficiency in streaming processing and face model recognition as well as deep model generation [19]–[21]. Additionally, the capacity of the cache, SSD, and HDD can be adjusted based on the application requirements and real budget. In our current design, the storage capacity is gradually increased to meet the needs of a large capacity of sensor pipe data and maintain a reasonable cost. To assist in the development of connected and autonomous vehicles and ensure the smooth communication between CAVs, edge servers, and remote clouds, *HydraSpace* will allocate a "transfer oriented space" as shown in Figure 1 to store the temperate data that is ready to be collected by the applications running on edge servers, road-side units, or clouds.

III. Implementation and Evaluation

In this section, we first present the initial implementation of *HydraSpace* and the devices we used in the evaluation. Then we show the results related to system performance, compression effects, and self-driving application performance as well as storage space reduction.

A. Initial Implementation

HydraSpace was implemented in the indoor mobile platform HydraOne [18] with the installation of six cameras and one 3D LiDAR for collecting sensor pipe

Device	No.	Summary
Computing System	1	32GB memory,Intel® Xeon(R) CPU E3-1275 v5 @ 3.60GHz × 8
Camera	6	SONY IMX322 CMOS
LIDAR	1	Velodyne Puck LITE, the default rotation speed is 600RPM and work frequency is 10 Hz

TABLE I
EXPERIMENT PARAMETERS

data. The specific parameters are presented in Table I. The camera model we used is based on SONY IMX322 CMOS with a frequency of 33Hz. It is connected via the USB port to the computing system. The LiDAR we used in our design is Velodyne Puck LITE, which is connected through the Ethernet with a default rotation speed of 600RPM and working frequency of 10 Hz. These devices were launched on top of the ROS [22]. The capacity of the cache, SSD and HDD for *HydraSpace* is set up as 16GB,250GB and 1TB, respectively. For the computation processes that we mentioned in Figure 1, we implemented adaptive compression algorithms for sensor pipe data, and we plan to add more computing progress in our future works.

The experiment is categorized into three steps as shown in Figure 2. First, we evaluate the system impact by utilizing six cameras and one 3D LiDAR. Second, we run different compression algorithms on image and point cloud to observe the compression effect. Third, we run various self-driving applications on the compressed data and compare their performance.

B. System Performance Evaluation

The impact on system performance is evaluated by capturing the memory percentage, CPU utilization, and power consumption with various combinations of the number of cameras and LiDAR. This is aimed at discovering the influence on the on-board computing system by varying the settings of cameras and LiDAR to find the optimal solution for the sensors. The experiment was carried out with four different settings: LiDAR only, LiDAR with two cameras, LiDAR with four cameras, and LiDAR with six cameras. The results are presented in Figure 3. We can see that the 6 cameras with LiDAR have the most impact on the system performance compared to other settings with an average of 35% CPU utilization and 0.3% memory usage as well as 12.8 Watts power consumption. We can also conclude that all four settings have a minimal influence on memory but a greater impact on CPU utilization with the increasing number of cameras. As it refers to power consumption, the LiDAR consumes 7.7 watts and the total power consumption would not be changed while the number of camera increases.

C. Compression Effects

The compression effect is evaluated by using compression ratio, compression time, and compression error. The compression ratio is defined as the ratio of the original data size to the compressed data size. The compression time refers to the total time spent on compressing a specific amount of the data. The compression error is defined as the deviation of the original data value and the decompressed data value. We run four different compression algorithms on image and point cloud data collected by the camera and LiDAR installed in the indoor platform to check the compression effects [23], [24]. We apply four different compression algorithms, including bzip2, lz4, sz, and zfp. The former two compression approaches are based on lossless algorithms while the latter two are based on lossy compression algorithms. Specifically, bzip2 is based on the Burrows Wheeler algorithm, which is one of the best overall compression algorithms. This method is capable of compressing files down to 15% or 10%. Lz4 provides extremely fast compression speed that is greater than 500MB/s per core. It encodes and decodes from a sliding window over previously seen characters, which results in both a smaller output and faster decompression times. As it refers to the lossy compression, sz is a fast error-bounded lossy compression solution that works effectively on large-scale HPC data sets [15]. It starts by linearizing multi-dimensional snapshot data and then predicts the successive data points with the best fit selection of curve fitting models. zfp is a lossy compressor for the integer and floating-point data stored in multidimensional arrays. By applying these compression algorithms to a vehicular data-set, we will have a complete understanding of the compression effects and find the best solution for on-board storage reduction. Figure 4 shows the compression outcomes on both point cloud and image datasets regarding compression ratio, compression time, and compression error among the algorithms mentioned above. We can see from the figure that the lossy compression method has better compressed size compared with the lossless algorithms for both point cloud and image datasets. Specifically, sz outperforms zfp due to its error-bounded selection and effective prediction of the data point with curve fitting models. Figures 5 and 6 illustrate the points and pixel variations among the four compression methods and their standard deviation, indicating the corresponding compression errors. After compression of the point cloud dataset using two lossless and two lossy algorithms, the results are presented in Figure 8. The results generated by the two lossless algorithms, bzip2 and lz4, have an apparent visual effect compared to the results produced by sz and zfp.

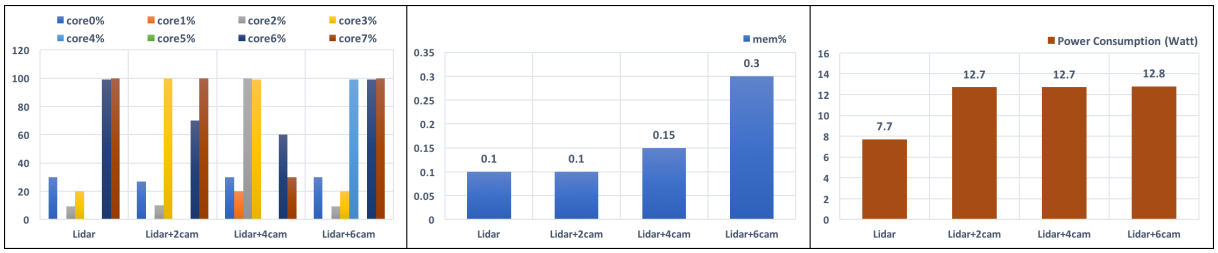
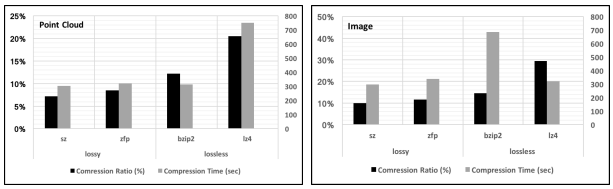


Fig. 3. System performance evaluation by running multiple sensors.



(a) (b)

Fig. 4. Compression performance among four methods on (a) point cloud (b) image.

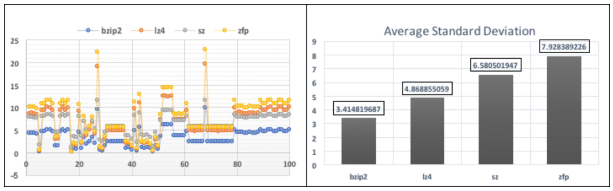


Fig. 5. Point cloud compression error among the 4 methods.

D. Self-driving Application Performance

After comparing the compression results using lossless and lossy algorithms, we need to conduct further investigation into the impacts on self-driving applications. Thus, we run an open-source TensorFlow Object Detection API built on top of TensorFlow to investigate the detection performance. We use images of varying quality as the input to construct, train, and deploy object detection models. The output pictures are presented in Figure 7. Figure 7(a) shows the object detection result of the original picture and its detection precision. Figures 7(b) and (c) show the results of the compressed image using sz and zfp lossy compression algorithms. We found that all these images have comparative detection performances with a precision of 98%, 95%, and 97%, respectively.

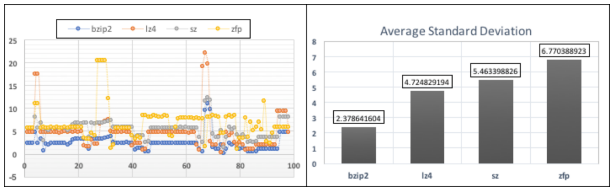


Fig. 6. Image compression error among the 4 methods.

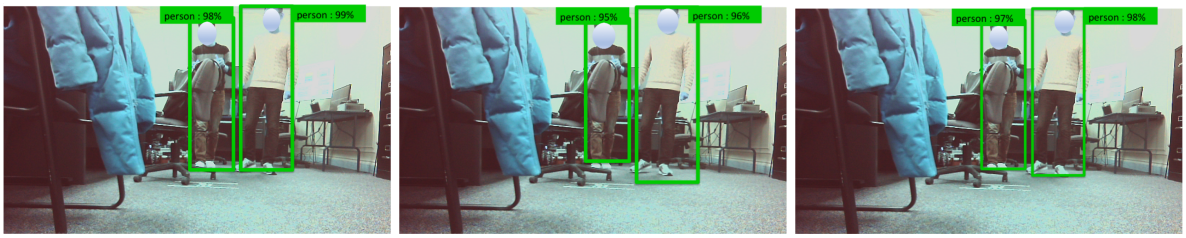
E. Storage Space Reduction

Based on the experiment results and observations, we propose an initial data management scheme for *HydraSpace* following the rules presented below. If the incoming source is a camera, the data will be streamed into the *HydraSpace* and compressed using the lossy compression algorithm sz, which has a faster compression time, accepted deviation, and comparative detection performance on a self-driving application. If it is from LiDAR, the point cloud data will be compressed by the bzip2 lossless algorithm due to the careful consideration of compression time and reconstruct performance as well as the scalability on the current system. Thus, we can save 88.6% of the space.

IV. Open Questions and Challenges

Autonomous driving is a complex task that relies on precise localization, navigation, planning, and system control. To maximize the safety and reliability of the vehicle, multiple sensors have been installed for collecting data and performing computing tasks. However, the amount of the sensed data is increasing rapidly, which far exceeds the capability and capacity of the existing on-board computing and storage systems.

The foremost challenge when developing a CAV storage system is that the architecture should cater to the different real-time application requirements, satisfy several computation needs, consider effective task offloading and data backup as well as provide fast communication between edge servers and remote clouds. Moreover, developers need to access and process sensed data, which is usually behind the CAN bus in a vehicle, as well as create and manage their applications by themselves. How to manage and share the data between the developers is still an open question. Many open source projects for data analytics are not reliable enough to meet the safety and reliability requirements; also, the lack of programming frameworks for accessing, processing and sharing CAV data among the VCU, road side servers,and



(a) (b) (c)

Fig. 7. Object detection results on (a) original image (b) compressed image using sz (c) compressed image using zfp.

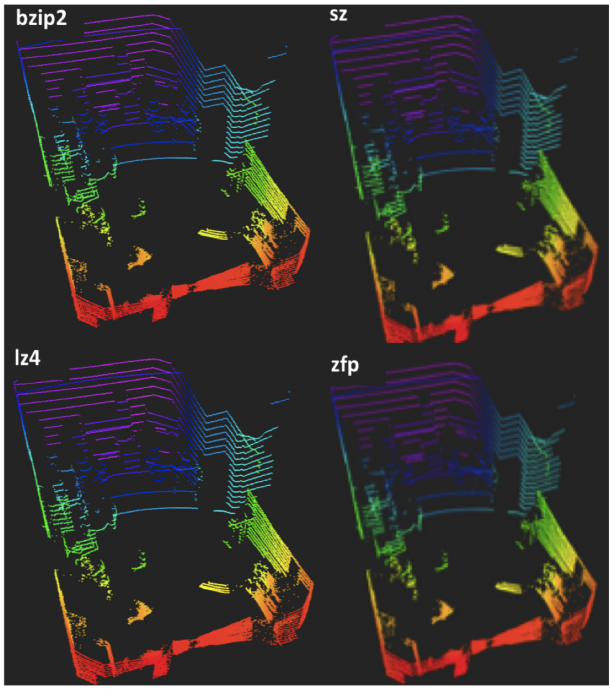


Fig. 8. Reconstruction of point cloud data among four methods.

the cloud [25] should be taken into consideration in the future development of CAVs.

Here we summarize five open questions and challenges for autonomous driving storage systems:

- **Synchronization:** Data on the autonomous driving vehicle has a variety of sources: its own sensors, other vehicles' sensors, RSU, and even social media. One big challenge to handle a variety of data sources is how to synchronize them. For example, a camera usually produces 30-60 frames per second while LiDAR's point cloud data frequency is 10HZ. For applications like 3D object detection which requires camera frames and point cloud at the same time, should the storage system do synchronization beforehand or let the application developer to do it? This issue becomes more challenging considering that the accuracy of the timestamps from different sensors fall into different granularity. For example, considering the vehicles that use network time protocol (NTP) for time synchronization, the timestamp difference can be as long as 100ms. For some sensors with a built-in GPS antenna the time accuracy goes to the nanosecond level, while other sensors get a timestamp from the host machine system time where the time accuracy is at the millisecond level. How to handle the sensor data with different frequency and timestamp accuracy is still an open question.
- **Privacy:** The sensors on a vehicle can capture much sensitive information, like people's faces, license plates, *etc.* In order to share the data without disclosure of personal privacy, how to mask privacy information from the raw data and maintain the whitelist/blacklist of access control has become a big challenge. For example, without proper access control methods, malicious applications might tamper this life-critical data, resulting in erroneous driving decisions and threatening the safety of passengers. Thus, how to securely access the data without any violations has become an urgent requirement for CAVs and their associated infrastructure. However, this is not an easy task due to the complexity of vehicular data, the high volume of access patterns as well as the lack of efficient control framework.
- **Security:** With rich data and a powerful computation platform, autonomous vehicles are supposed to provide a service that allows outside developers to run applications on it [25]. The storage system provides not only raw data but also computing resources for developers to access. How to leverage Trusted Execution Environment (TEE) to run applications remains an open issue. Specifically, this is a two-way security challenge. On the one hand, a question is how to prevent the host (vehicle) from knowing the function executed by third parties, such as searching for a person wanted by law enforcement. On the other hand, another question is how to prevent third party applications from attacking the host (vehicle) and other applications running on the

vehicle. Both require further investigation.

- **Data movement:** Data movement happens everywhere since the whole autonomous driving pipeline is built on sensing and perception. Several running applications/processes may require the same data, and several pieces of data may be needed for the execution of one application. Usually a middleware system is used as the middle-level between applications and hardware/software infrastructure for efficient data movement, like Robot Operating System (ROS) [22] and Cyber [26]. However, how to share data among processes or vehicles with low latency and less overhead is still an open issue.
- **Reliability and redundancy:** To make data sharing a service, guaranteeing reliability is an important issue. This problem becomes even more severe when considering the limited storage space and the huge amount of data sensors can generate. In general, for a distributed system, several replicas are maintained on several nodes to ensure reliability. The first question is how many replicas are necessary and where to store these replicas. For the vehicle scenario, the storage on an RSU is a potential option for the vehicle to store replicas of their data. The next question is how to ensure the consistency considering the unreliable communications between the vehicle and RSU [27].

V. Related Work

Ratnasamy *et al.* developed a Geographic Hash Table system to make effective use of the vast amounts of data gathered by large-scale sensor networks [28]. The Geographic Hash Table system (GHT) hashes keys into geographic coordinates and stores a key-value pair at the sensor node geographically nearest to the hash of its key. The system replicates stored data locally to ensure persistence when nodes fail. It also distributes the load throughout the network using a geographic hierarchy. The results demonstrate that GHT is the preferable approach as it offers high data availability and scales to large sensor-net deployments even when nodes fail or are mobile. Zeng *et al.* proposed a layered architecture of cloud storage and discussed the key technologies involving deployment, storage virtualization, data organization, migration, security, *etc.* [29]. The operation mechanism includes an ecology chain, game theory, ant colony optimization, data life cycle management, maintenance and update, convergence and evolution mechanisms. Gibson *et al.* [30] proposed the Network-Attached Secure Disk (NASD) storage architecture, which provides scalable storage bandwidth without the cost of servers used primarily for transferring data from peripheral networks. Huston *et al.* [31] proposed a storage architecture for early discard for an interactive search of unindexed data named Diamond. The system optimizes the evaluation order of the filters based on the run-time measurements of each filter's selectivity and computational cost. Diamond can also dynamically partition the computation between the storage devices and the host computer to adjust for changes in hardware and network conditions. Performance numbers show that Diamond dynamically adapts to a query and run-time system state.

For compression techniques, image compression algorithms can be divided into two categories: lossless compression and lossy compression. Lossless compression ensures that the reconstructed image is exactly the same as the original image, but its compression ratio and efficiency are inferior to lossy compression. Lossy compression does not restore the original data completely, but the information loss has little impact on the understanding of the original image, resulting in a much larger compression ratio. Currently, there are two main kinds of lossy compression methods to maintain the color image. First are direct methods. This way is to compress the image directly in the spatial domain, using techniques that mainly consist of block truncation [32] and vector quantization [33]. Second are transform methods. This way uses discrete cosine transform (DCT) [34], discrete wavelet transform (DWT) [35], principal component analysis (PCA) [36], and other transformations to process the image in a transform domain. The purpose of these transformations is to concentrate the energy of the original image into less transform coefficients.

VI. Summary

With the explosive growth of data collected by multiple sensors, data storage is the critical factor for the future of Autonomous Vehicles. Unfortunately, there has been no solid and comprehensive research conducted in this field. Therefore, we propose a computational storage *HydraSpace* with multi-layered storage architecture that adopts effective compression algorithms to manage the sensor pipe data. We also conduct a comprehensive experiment on the indoor experimental platform to study the system performance regarding CPU and memory utilization, power consumption for different sensors, as well as the data-set compression with the consideration of different data types. This would be extremely helpful for the application development and system optimization for future autonomous vehicles. For future work, we will focus on the evaluation of various computing platforms, reduce the data stored in *HydraSpace* by incorporating the feature extraction of the collected data, and implement more computation processes. In addition, five open questions and challenges have been discussed to envision the future development of storage for autonomous vehicles.

REFERENCES

[1] Flood of data will get generated in autonomous cars. https://autotechreview.com/features/flood-of-data-will-get-generated-in-autonomous-cars. Accessed: 2020-2-18.

[2] Data storage is the key to autonomous vehicles' future. https://iotnowtransport.com/2019/02/12/71015-data-storage-key-autonomous-vehicles-future/. Accessed: 2019-12-30.

[3] The basics of LiDAR - light detection and ranging - remote sensing. https://www.neonscience.org/lidar-basics. Accessed: 2020-2-18.

[4] S. Ilic, J. Katupitiya, and M. Tordon. In-vehicle data logging system for fatigue analysis of drive shaft. In *International Workshop on Robot Sensing, 2004. ROSE 2004.*, pages 30–34, 2004.

[5] Yunsheng Wang, Holger Weinacker, and Barbara Koch. A LiDAR point cloud based procedure for vertical canopy structure analysis and 3d single tree modelling in forest. *Sensors*, 8(6):3938–3951, Jun 2008.

[6] D. Steinhauser, O. Ruepp, and D. Burschka. Motion segmentation and scene classification from 3d LiDAR data. In *2008 IEEE Intelligent Vehicles Symposium*, pages 398–403, June 2008.

[7] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2922–2930. JMLR.org, 2017.

[8] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[9] Nanrun Zhou, Haolin Li, Di Wang, Shumin Pan, and Zhihong Zhou. Image compression and encryption scheme based on 2d compressive sensing and fractional mellin transform. *Optics Communications*, 343:10 – 21, 2015.

[10] Y. Zhou, C. Wang, and X. Zhou. DCT-based color image compression algorithm using an efficient lossless encoder. In *2018 14th IEEE International Conference on Signal Processing (ICSP)*, pages 450–454, Aug 2018.

[11] Eli Shusterman, Meir Feder, and Senior Member. Image compression via improved quadtree decomposition algorithms, 1994.

[12] Muhammad Ali Qureshi and M. Deriche. A new wavelet based efficient image compression algorithm using compressive sensing. *Multimedia Tools and Applications*, 75(12):6737–6754, Jun 2016.

[13] Hanaa ZainEldin, Mostafa A. Elhosseini, and Hesham A. Ali. Image compression algorithms in wireless multimedia sensor networks: A survey. *Ain Shams Engineering Journal*, 6(2):481 – 490, 2015.

[14] X. Peng, J. Hou, L. Tan, J. Chen, J. Jiang, and X. Guo. Bit-Error aware lossless color image compression. In *2019 IEEE International Conference on Electro Information Technology (EIT)*, pages 126–131, May 2019.

[15] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 730–739, May 2016.

[16] James Diffenderfer, Alyson Fox, Jeffrey Hittinger, Geoffrey Sanders, and Peter Lindstrom. Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing*, 41:A1867–A1898, 02 2019.

[17] Chiranjeevi Karri and Umaranjan Jena. Fast vector quantization using a bat algorithm for image compression. *Engineering Science and Technology, an International Journal*, 19(2):769 – 781, 2016.

[18] Yifan Wang, Liangkai Liu, Xingzhou Zhang, and Weisong Shi. HydraOne: An indoor experimental research and education platform for CAVs. In *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[19] Gennady Pekhimenko, Chuanxiong Guo, Myeongjae Jeon, Peng Huang, and Lidong Zhou. Tersecades: Efficient data compression in stream processing. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, page 307–320, USA, 2018. USENIX Association.

[20] Ping Luo, Zhenyao Zhu, Ziwei Liu, Xiaogang Wang, and Xiaoou Tang. Face model compression by distilling knowledge from neurons. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 3560–3566. AAAI Press, 2016.

[21] Bharat Bhusan Sau and Vineeth N. Balasubramanian. Deep model compression: Distilling knowledge from noisy teachers. *CoRR*, abs/1610.09650, 2016.

[22] Robotic operating system. https://www.ros.org/. Accessed: 2020-2-18.

[23] Junguo Zhang, Qiumin Xiang, Yaguang Yin, Chen Chen, and Xin Luo. Adaptive compressed sensing for wireless image sensor networks. *Multimedia Tools and Applications*, 76(3):4227–4242, Feb 2017.

[24] D. Minnen, G. Toderici, M. Covell, T. Chinen, N. Johnston, J. Shor, S. J. Hwang, D. Vincent, and S. Singh. Spatially adaptive image compression using a tiled deep network. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2796–2800, Sep. 2017.

[25] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong. Openvdap: An open vehicular data analytics platform for cavs. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1310–1320, 2018.

[26] Baidu. Apollo Cyber. [Online]. https://github.com/ApolloAuto/apollo/tree/master/cyber.

[27] Liangkai Liu, Baofu Wu, and Weisong Shi. A comparison of communication mechanisms in vehicular edge computing. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, June 2020.

[28] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. Ght: A geographic hash table for data-centric storage. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, page 78–87, New York, NY, USA, 2002. Association for Computing Machinery.

[29] Wenying Zeng, Yuelong Zhao, Kairi Ou, and Wei Song. Research on cloud storage architecture and key technologies. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, page 1044–1048, New York, NY, USA, 2009. Association for Computing Machinery.

[30] Garth A. Gibson, David F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim Zelenka. A cost-effective, high-bandwidth storage architecture. 32(5), 1998.

[31] Larry Huston, Rahul Sukthankar, Rajiv Wickremesinghe, Mahadev Satyanarayanan, Gregory R Ganger, Erik Riedel, and Anastassia Ailamaki. Diamond: A storage architecture for early discard in interactive search. In *FAST*, volume 4, pages 73–86, 2004.

[32] E. Delp and O. Mitchell. Image compression using block truncation coding. *IEEE Transactions on Communications*, 27(9):1335–1342, 1979.

[33] N. M. Nasrabadi and R. A. King. Image coding using vector quantization: a review. *IEEE Transactions on Communications*, 36(8):957–971, 1988.

[34] Wen-Hsiung Chen, C. Smith, and S. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on Communications*, 25(9):1004–1009, 1977.

[35] M. J. Shensa. The discrete wavelet transform: wedding the a trous and mallat algorithms. *IEEE Transactions on Signal Processing*, 40(10):2464–2482, 1992.

[36] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.