2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC) | 978-1-6654-2463-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/COMPSAC51774.2021.00017

Streaming Data Priority Scheduling Framework for Autonomous Driving by Edge

Lingbing Yao
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
201821034002@mail.scut.edu.cn

Hang Zhao
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
i@hang.im

Jie Tang
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
cstangjie@scut.edu.cn

Shaoshan Liu
PerceptIn
Fremont, United States
shaoshan.liu@perceptin.io

Jean-Luc Gaudiot
Dept. of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, United States
gaudiot@uci.edu

***Abstract*—In recent years, intelligent vehicles like autonomous vehicles generate a huge amount of sensing data continuously. The computations on those data streams are far beyond the processing capacity of on-board computing. To deal with the streaming data process in real-time, the deployment of streaming data processing system by edge turns to the first choice in terms of performance. However, the existing frameworks cannot satisfy the complicated demands from autonomous driving tasks and lack the ability in supporting the task priority scheduling. In this paper, we propose a streaming data priority scheduling framework for autonomous driving by edge on Spark Streaming and make an implementation on Spark 2.3.0. The proposed framework can identify the priorities among different data processing tasks and implement the task scheduling based on non-preemptive priority queuing theory. To meet differentiated service level requirements, the proposed non-preemptive priority queuing scheduling mechanism considers the priority category of tasks, the distance between vehicles and edge nodes, and the priority weight of vehicles. Experiments show that this mechanism can effectively identify the priority information of different tasks from different vehicles and reduce the end-to-end latency of high-priority tasks by up to 46% than low-priority tasks.**

Keywords*—*Autonomous driving, Streaming data processing, Priority scheduling, Spark Streaming

I. INTRODUCTION

In recent years, modern vehicles have become increasingly intelligent and lots of Advanced Driving Assistance technology even autonomous driving begins the large-scale application. These autonomous vehicles (AVs) deploy kinds of sensors, like Radar, Camera, and Lidar for the full pipeline of on-vehicle intelligence. These sensors are used to sense the environment, output the captured data continuously, and work for the sensing/ perception/decision of autonomous driving. For instance, Tesla's Autopilot adopts 8 cameras and 12 ultrasonic sensors to achieve L2-level automatic assisted driving [1]. However, these streaming sensor data are generated at a tremendous rate. If all these sensors are in a state of work, the total data rate can be up to 750Mbps. If working 8 hours per day, some automatic vehicle of hybrid scheme could generate about up to 2,700GB data.

The consequent huge amount of streaming data process demands has far exceeded the computing capacity of on-board computing devices. Therefore, researchers begin to employ the distant cloud computing data-center and nearby edge computing to take care of streaming data processing for autonomous driving. However, the remote cloud inevitably brings about two problems: First, massive streaming sensor data will be uploaded at the same time, which may cause significant stress on the existing bandwidth and lead to an unstable data transmission; Second, for some delay-sensitive tasks like SLAM, the instability and the high delay in data processing will put the passengers and surrounding vehicles or pedestrians in danger.

Compared to the remote cloud, edge computing is deployed closer to the data source, the on-board sensors [2]. With the assistance of the computing and storage capacity by edge, the massive streaming data of AVs can be processed in a more real-time and low latency fashion. However, the existing streaming data process frameworks are designed without considering the deadline requirements and priority differences between tasks, thus they are not engaging with a differentiated priority scheduling pattern. Meanwhile, with the prosperity of on-vehicle application ecosystem, rich types of tasks tend to have kinds of priority demands on the data processing deadline, payment, and resource sensitivity. The existing frameworks meet great difficulties in processing streaming data from various applications especially in multi-vehicle scenario.

To this end, in this paper we proposed a streaming data priority scheduling mechanism and build a corresponding streaming data processing framework by edge for autonomous driving (Sec. III). This framework can identify the categories of data processing tasks from multi-vehicles, generate the corresponding priority information and conduct the task scheduling based on the non-preemptive priority queuing theory. The non-preemptive priority queuing scheduling mechanism fully considers the information like the priority category of tasks, the distance between vehicles and edges, and the priority weight of vehicles. This mechanism is expected to make full use of the limited computing and storage resources by edge to provide low latency, high throughput, and task differentiated processing

978-1-6654-2463-9/21/$31.00 ©2021 IEEE
DOI 10.1109/COMPSAC51774.2021.00017

capability for large-scale and extensive deployment of autonomous driving in the future. We also implemented the proposed mechanism in Spark Streaming. and present out the prototype in Sec. VI. Its evaluation in real scenarios in the environment of Ubuntu 18.04 LTS is stated in Sec. VII.

The contribution of this paper includes:

- We have proposed a streaming data priority scheduling framework by edge for autonomous driving. It is designed to enable the identification and scheduling of prioritized kinds of streaming data process by edge.
- We have modeled the multi-task scheduling based on non-preemptive priority queuing theory and designed a non-preemptive priority queuing scheduling mechanism to meet the requirements of differentiated service.
- We implemented a streaming data processing framework prototype in Spark 2.3.0. The experimental results reveal that this mechanism can effectively improve the data processing ability of edges. The end-to-end latency of high-priority tasks was reduced by 46% than low-priority tasks, thus it is proved to offer a differentiated service provision for autonomous driving task processing.

II. Background and Motivation

A. Spark and Spark Streaming

Spark is a distributed big data processing framework [3], whose core abstraction is resilient distributed dataset (RDD) that can be partitioned across the cluster nodes. A Spark application runs on a cluster (including a driver program and several executors). When running, the driver program defines a directed acyclic graph (DAG) of RDDs for the job and then Spark's scheduler splits the DAG into stages. Next, tasks are assigned to available executors for computation and storage.

Spark Streaming [4], as an extension of Spark, is a real-time stream processing framework, whose key abstraction is Discretized Stream (DStream), which is built on RDD. The core idea of DStream is to split the real-time input data stream into several small *batch*es according to an assigned period time called the *batch interval*. Then, batches are delivered to Spark's execution engine for processing. Finally, the result streaming data are produced, which are also organized by batches.

B. Motivations

Considering the application background of the proposed streaming data processing framework by edge, which has high requirements for burst traffic, system stability, and fault recovery capability, we consequently choose Spark Streaming as the implementation basis and make modifications to it.

We notice that the existing streaming data processing frameworks are mainly working for cloud computing and fail to fully consider the features of data processing under the edge computing environment. They have common limitations that: 1) the low resource utilization rate and 2) no priority service differentiation, etc. When faced with complex tasks of AVs, the existing frameworks encounter difficulty in meeting the diversified level of service demands of AVs.

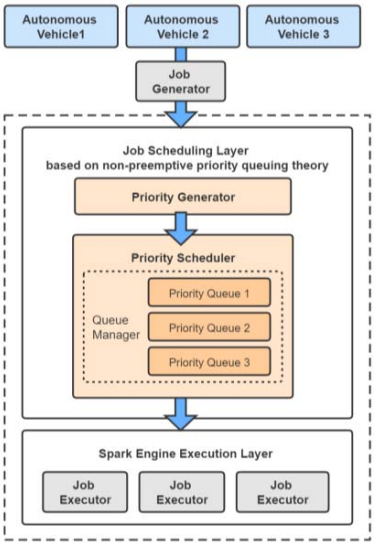


Fig. 1. The architecture of the proposed framework and the workflow of our job scheduling mechanism.

III. System Overview

The proposed framework is deployed in edge nodes and regards the streaming data generated by AVs in a specific area, whose architecture is as shown in Fig. 1. The framework contains two layers: 1) the top layer, the job scheduling layer based on non-preemptive priority queuing theory, and 2) the bottom layer, the Spark engine execution layer.

In this study, we mainly focus on the design and implementation of the job scheduling layer, which contains a *Priority Generator* for job priority generation and a *Priority Scheduler* for job priority scheduling. After the received data are divided into different batches according to the batch interval, the streaming job is generated and submitted to the *Job Scheduler* for scheduling. The *Job Priority Generator* in the *Job Scheduler* first scans the data in the batch, calculates the priority based on the rules, and then assigns priority to the job. Priority *Job*s are submitted to the corresponding *Priority Queue* under the management of the *Queue Manager*. When a *Job Executor* has freed the computing resources, the *Job Scheduler* takes the job with the highest priority from the *Priority Queue* and commits it to execution.

To apply this streaming data priority scheduling mechanism, we implement a corresponding streaming data processing framework in virtue of Spark Streaming, which can fully satisfy the streaming data processing requirements of AVs.

IV. Task Priority Calculation

Spark Streaming utilizes the First Come First Service (FCFS) scheduling strategy for the received jobs by default. Considering that our scenario of this study is autonomous driving, different tasks may have different demands for priority.

When autonomous driving are promoted commercially, for different vehicles, due to their different distances from base stations, different payment levels of passengers and different

types of tasks to be processed, the uploaded data will also be given different priorities. We assume that the data record uploaded by an autonomous car to the Spark Streaming system contains a group of metadata, including the car information and on-board task information as shown in Table I and Table II.

As a streaming data processing framework for autonomous driving, it should be able to distinguish the urgency and priority of tasks among different AVs through this metadata information, so that it can satisfy the urgent needs of high-priority vehicles or high-priority delay-sensitive tasks. Assuming that we extract the priority information needed: <*c_prio, c_time, t_prio, t_time*>, and the priority of a record is defined as:

$$p = A \times c\_prio + B \times t\_prio - \min\{C \times c\_time, D \times t\_time\} \quad (1)$$

We set *c_prio* and *t_prio* to be 1 or 2 respectively, and let $A = 100$, $B = 100$, $C = 1$, and $D = 1$. We constraint the lower bound of priority p to be 0 and then p is in the range of [0, 400]. Besides, a greater p indicates a higher priority. We define the priority of a batch to be the average priority of the records within the batch.

When a job is scanned by *Priority Generator* and assigned a priority, to facilitate the subsequent establishment of the priority queue, we map the numerical priorities to five levels in descending order as shown in Table III. Note that the high-priority jobs run earlier than the low-priority jobs.

TABLE I. THE CAR INFORMATION

	Parameters	Note
c_id	Car ID	
c_type	Car type	
c_prio	Car priority	Two levels: high/low, corresponding to 2/1.
c_dir	Driving direction	Two directions: approaching/leaving.
c_dist	Distance from the base station	Unit: meter (m).
c_speed	Driving speed	Unit: kilometer per hour (km/h).
c_time	The remaining time of the base station coverage signal	Estimated by driving direction, driving speed, and distance from the base station. Unit: second(s).

TABLE II. THE ON-BOARD TASK INFORMATION

	Parameters	Note
t_id	Task ID	
t_type	Task type	
t_prio	Task priority	Two levels: high/low, corresponding to 2/1.
t_time	The required time to complete the task	Unit: second (s).

TABLE III. QUANTIFICATION OF PRIORITIES

Priority Level	Average Priority Range of Batch
VERY_HIGH	[320,400]
HIGH	[240,320)
NORMAL	[160,240)
LOW	[80, 160)
VERY_LOW	[0, 80)

V. NON-PREEMPTIVE PRIORITY SCHEDULING

A. Non-preemptive Priority Queuing Theory Modeling

We introduce the queuing theory model to the job scheduling system of Spark Streaming, in which the model used is $M_1/M_2/S/\infty$. M_1 indicates that the successive arrival rate of the job follows the negative exponential distribution with parameter λ, where λ is the average number of arrived jobs per unit time; M_2 is the distribution of the service rate follows the negative exponential distribution with parameter μ, where μ is the average number of jobs completed by the service per unit time; S is the number of resource pool service windows; ∞ represents unlimited system capacity. If Spark Streaming is regarded as a random service system, the randomly arrived job is similar to the customer in the queuing theory model, and the queue is the computing slot in the scheduling system.

In this section, we discuss the FCFS scheduling scheme and the non-preemptive priority-based scheduling scheme in turn. To simplify the analysis, we only discuss the case of $S = 1$.

a) Derivation of M/M/1 Model (FCFS): When the FCFS scheduling scheme is adopted, jobs are queued in the queue according to their arrival order and receive services in turn. We define the service strength as $\rho = \frac{\lambda}{\mu}$. If the service strength is greater than 1, it means that the number of arrived jobs per unit time is greater than the number of jobs that completed the service, and the length of the queue in the system is getting longer and longer, which causes system blocking. When $0 < \rho < 1$, according to related principles of queuing theory, it is easy to obtain the line length $L_S = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$, the queue length $L_q = \frac{\rho\lambda}{\mu-\lambda}$, the stay time $W_S = \frac{L_S}{\lambda}$, and the queue time $W_q = \frac{L_q}{\lambda}$.

b) Derivation of Non-preemptive Priority M/M/1 Model: In the priority queuing model, jobs are graded. Suppose that tasks are divided into N levels ($N = 5$), the first level has the highest priority, and the Nth level has the lowest priority. For the non-preemptive priority service model of $M/M/1$ queuing theory, let λ_i be the arrival rate of the job at level i, and then the arrival rate of the system is $\lambda = \sum_{i=1}^{N} \lambda_i$. μ_i is the average service rate of the job at level i. S_i is the service time required for the job at level i. S is the system average service time.

Let $\rho_i = \frac{\lambda_i}{\mu_i} = \lambda_i * S_i (1 \le i \le N)$ be the traffic of the priority i, and $\rho = \sum_{i=1}^{N} \rho_i = \sum_{i=1}^{N} \lambda_i S_i$ be the traffic of the system. Since the job at level 1 to N are all independent Poisson flows, the probability that the job reaches the system at any time belongs to level i is $\frac{\lambda_i}{\lambda}$.

c) The First-Priority Job: When the newly arrived job is of the first priority, its average waiting time in the system is composed of two parts:

One part is the sum of the average service time T_1 of all the first-priority jobs that are waiting in line for service. Note that the average number of queued jobs is L_{q1}, then:

$$T_1 = S * L_{q1} = \frac{L_{q1}}{\mu} = \rho_1 * W_{q1} \quad (2)$$

Where W_{q1} is the average queue waiting time.

The other part is the average time to wait for the service desk to be vacated. Due to the memory lessness of the negative exponential function, it should be

$$T_2 = \rho * \frac{1}{\mu} = \frac{\lambda}{\mu^2} \tag{3}$$

Then, we get:

$$W_{q1} = \rho_1 * W_{q1} + \frac{\lambda}{\mu^2} \tag{4}$$

So, the average waiting time is:

$$W_{q1} = \frac{\lambda}{\mu^2(1-\rho_1)} = \frac{\lambda}{\mu(\mu-\lambda_1)} \tag{5}$$

According to Little's Law:

$$L_s = \lambda * W_s, L_q = \lambda * W_q, L_s = L_q + \rho, W_s = W_q + \frac{1}{\mu} \tag{6}$$

The first-priority job, namely the job of VERY_HIGH priority, has the average length of stay in the system of:

$$W_{s1} = W_{q1} + \frac{1}{\mu} = \frac{\mu + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5}{\mu(\mu-\lambda_1)} \tag{7}$$

The average waiting queue length is:

$$L_{q1} = \lambda_1 * W_{q1} = \frac{\lambda * \lambda_1}{\mu(\mu-\lambda_1)} \tag{8}$$

The average line length is:

$$L_{s1} = \lambda_1 * W_{s1} = \frac{\lambda_1(\mu + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5)}{\mu - (\mu - \lambda_1)} \tag{9}$$

d) The Second-priority Job: When the newly arrived job is of the second priority, its average waiting time in the system is composed of three parts:

The first part is the sum of the average service time T_1 of the first-priority and second-priority jobs waiting in line for service:

$$T_1 = \rho_1 W_{q1} + \rho_2 W_{q2} \tag{10}$$

The second part is the average time T_2 waiting for the service desk to be vacated:

$$T_2 = \frac{\lambda}{\mu^2} \tag{11}$$

The third part is the sum of the average delay time T_3 caused by the priority interruption of the first-priority job successively arriving during the queue of the new second-priority job:

$$T_3 = \rho_1 W_{q2} \tag{12}$$

Accumulating the above three formulas, we get:

$$W_{q2} = \rho_1 W_{q1} + \rho_2 W_{q2} + \frac{\lambda}{\mu^2} + \rho_1 W_{q2} \tag{13}$$

Then we get the average waiting time:

$$W_{q2} = \frac{\rho_1 W_{q1} + \frac{\lambda}{\mu^2}}{1-\rho_1-\rho_2} = \frac{\lambda}{(\mu-\lambda_1)(\mu-\lambda_1-\lambda_2)} \tag{14}$$

According to Little's Law, the average stay time of the second-priority job in the system is:

$$W_{s2} = W_{q2} + \frac{1}{\mu} \tag{15}$$

The average waiting queue length is:

$$L_{q2} = \lambda_2 W_{q2} = \frac{\lambda\lambda_2}{(\mu-\lambda_1)(\mu-\lambda_1-\lambda_2)} \tag{16}$$

The average line length (the waiting jobs and the jobs being served by the service desk) is:

$$L_{s2} = \lambda_2 W_{s2} \tag{17}$$

e) The Third/Fourth/Fifth-priority Job: Similarly, the average waiting time of the third/fourth/fifth-priority customers can be obtained:

$$W_{q3} = \frac{\lambda}{(\mu-\lambda_1-\lambda_2)(\mu-\lambda_1-\lambda_2-\lambda_3)} \tag{18}$$

$$W_{q4} = \frac{\lambda}{(\mu-\lambda_1-\lambda_2-\lambda_3)(\mu-\lambda_1-\lambda_2-\lambda_3-\lambda_4)} \tag{19}$$

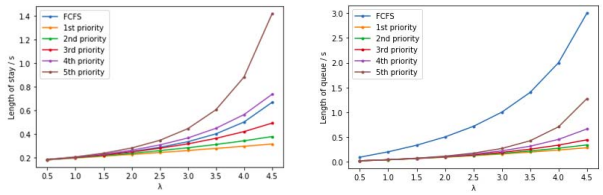
$$W_{q5} = \frac{\lambda}{(\mu-\lambda_1-\lambda_2-\lambda_3-\lambda_4)(\mu-\lambda_1-\lambda_2-\lambda_3-\lambda_4-\lambda_5)} \tag{20}$$

According to Little's Law, we can get the other three running indicators of the third/fourth/fifth-priority job.

B. Numerical Analysis

Based on the theoretical derivation, we can evaluate the performance of the queuing model using FCFS and priority scheduling through numerical analysis.

Assume that the Spark Streaming system processes multiple jobs from multiple *Receivers*. The system receives 5 jobs per



(a) (b)

Fig. 2. The comparison of (a) average stay time and (b) average line length.

TABLE IV. RUNNING RESULT AT $\mu = 6, \lambda = 5$

Type	Waiting time/min	Stay time/min	Waiting line length	Line length
FCFS	0.833	1	4.167	5
The 1st priority	0.167	0.333	0.167	0.333
The 2nd priority	0.250	0.417	0.250	0.417
The 3rd priority	0.417	0.583	0.417	0.583
The 4th priority	0.833	1	0.833	1
The 5th priority	2.500	2.667	2.500	2.667

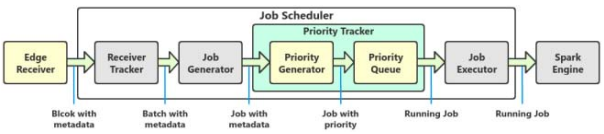


Fig. 3. Execution flow of job scheduling layer based on non-preemptive priority queuing theory.

minute and each priority job has the same frequency. At the same time, the system can complete the processing of 6 jobs per minute. Then we can get the available model parameters $\mu = 6$ and $\lambda_1 = \lambda_2 = \cdots = \lambda_6 = 1$. According to the above parameters, the running indicators of each priority in the FCFS queue and the non-preemptive priority queue are shown in Table IV. It can be observed from Table IV that the average stay time of the first-priority and second-priority jobs has decreased by 58.3% and 66.7% respectively than the FCFS queue. The average line length is much smaller than the FCFS queue, meeting the needs of delay-sensitive tasks. Although the average stay time of the fifth-priority job has increased by 166.7% relative to the FCFS queue, considering that its priority is lower, it is within the acceptable range of the system.

Let $\mu = 6$ and we get the comparison of the average stay time and average line length as Fig. 2 by changing λ.

VI. System Implementation

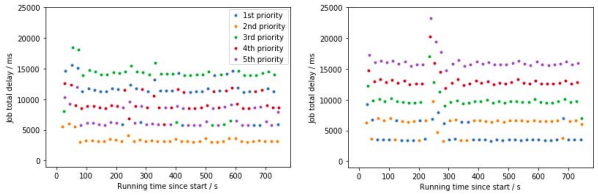
The proposed framework is divided into two levels from top to bottom: the job scheduling layer based on the non-preemptive priority queuing theory, and the Spark engine execution layer. We implement the top layer by modifying the source code of Spark Streaming and directly use Spark Engine as the bottom. Relying on the existing interface, we mainly realize two parts of the top: *Edge Receiver* for user-defined data receiving; and *Priority Tracker* for priority management, including *Priority Generator* and *Priority Queue*.

Fig. 3 shows the execution flow of the job scheduling layer. First, *Edge Receiver* receives the sensing data, which contains metadata. Second, *Edge Receiver* gathers together the sensing data with metadata, divides them into blocks, and then inputs them into *Receiver Tracker*. Third, after batch generation and job generation in sequence, a corresponding job is created and becomes the input of the *Priority Tracker* module. Next, the *Priority Tracker* identifies the job it receives, checks the block and batch information, and extracts the corresponding metadata. Then, *Priority Tracker* conducts metadata processing to generate the corresponding priority information, Finally, the jobs with priorities are added to the *Priority Queue* and then executed in order of priorities from highest to lowest.

VII. Experimental Evaluation

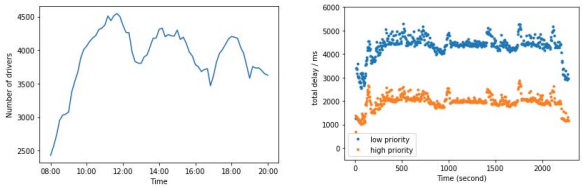
A. Experiment Setup

We implement the streaming data priority scheduling mechanism and conduct corresponding experiments. Our test server is Intel(R) Xeon(R) CPU E5-2609 v4@1.70GHz with 32GB RAM and 4TB HDD hard disk. The operating system is



(a) (b)

Fig. 4. The total delay of jobs with different priorities under (a) FCFS and (b) priority scheduling, respectively.



(a) (b)

Fig. 5. (a) The traffic flow of one day in the real scenario; (b) Comparison of the end-to-end latency for tasks with different priorities.

Ubuntu 18.04 LTS, and the software environment is built on Spark 2.3.0, which relies on JDK 1.8 and Scala 2.11.12.

B. Non-preemptive Priority Queuing Theory Scheduling

To verify the scheduling effect, we construct an application containing 5 data sources. Each application corresponds to different priority data. Each data source has the same data arrival rate. We set the number of *Receiver* nodes to 5 and the number of *Executor* nodes to 1, and observe the execution of jobs corresponding to data sources with different priorities in the same batch. To facilitate observation, we set the batch interval to 20s, and turn off the dynamic adjustment function of batch interval. Fig. 4(a) shows the execution result of different priority jobs when using FCFS. In Fig. 4(a), the abscissa represents the running time since the application start, and the ordinate represents the time from job creation to execution completion. In Spark Streaming, the sequence of job generation has a certain regularity (the order of occurrence in user code), so jobs will be executed in a specific order. However, this order does not conform to the user's expected priority, which will cause high-priority jobs to be executed after low-priority jobs. Fig. 4(b) shows the execution of different priority jobs after enabling non-preemptive priority scheduling. At this time, the system schedules and executes jobs according to the priority of jobs, and jobs with higher priorities are given priority execution, effectively reducing the end-to-end delay of high-priority jobs.

C. Comprehensive Experiment

To examine the performance of this framework in the real scenario, we test it based on the real traffic flow data about the online-hailing cars of Didi Chuxing [5]. We extract the traffic flow from 08:00 to 20:00 on Oct.1, 2016 in Fig. 5(a) as the input data rat. At the same time, we set two task priorities of high and low, and make their quantity ratio to 1:2. Fig. 5(b) shows the latency of tasks with different priorities. We observe that a high-priority task is always executed before a lower-priority task. The average end-to-end delay of low priority is 5279ms, while that of a high-priority task is only 2862ms. Through our priority scheduling, the average end-to-end delay of high-priority tasks is reduced by 46% compared to low priority tasks.

VIII. Related Work

In Hadoop, Q. Sun introduced the M/G/1 queuing model into the job scheduling system and proposed a non-preemptive short-job-first scheduling scheme, which effectively shortened the average waiting time of all jobs [6]. L. Gu proposed an improved priority scheduling algorithm, which considered the computing capacity of cluster nodes and the overall system load level, and assigned tasks according to the job priority and the computing capacity of nodes, ensuring the overall system load falling into a reasonable range [7].

In Spark, K. Liu [8] studied the task scheduling strategy in a heterogeneous environment and proposed a scheduling scheme of assigning executing nodes for tasks based on node performance and task complexity. J. Tang et al. [9] proposed a DAG reuse scheme based on the genetic algorithm, which sorted the execution order of DAG to reduce the total execution time. Islam et al. [10] implemented a cost-efficient scheduling strategy by centralizing the configuration of Executor as much as possible to reduce the number of virtual machines (VMs) so that it can reduce costs while satisfying task deadlines.

In Spark Streaming, T. Ajila et.al [11] proposed data-driven priority scheduling for text processing tasks with priority, which generates priority information by pre-scanning the input data. R. Birke et al. [12] proposed Dslash, a latency-driven data control strategy, which ensures the controllability of system delay by adjusting the maximum data capacity of block. H. Jin et al. [13] adopted the pre-scheduling method to identify potential stragglers in Spark Streaming in advance, and they improved the system resource utilization through reasonable pre-assignment of tasks. Reference [14] proposed A-scheduler, which divides tasks into data-dependent tasks and data-independent tasks, and places them in two task queues for scheduling. Meanwhile, they adopted reinforcement learning to adaptively adjust the parallelism and weight of tasks.

IX. Conclusion

With the development of autonomous driving, on-board sensors generate massive streaming data. Existing streaming data processing frameworks based on cloud computing have shortcomings of insufficient bandwidth and high transmission delay, which are difficult to satisfy the demands of stability and low latency for autonomous driving tasks. In contrast to clouds, edge computing, which is limited in scale of computing and storage capabilities, is more suitable for the processing of delay-sensitive tasks for autonomous driving.

To satisfy the diverse task processing requirements of AVs, we propose a streaming data priority scheduling mechanism by edge and implement a streaming data processing framework on Spark Streaming. It can distinguish the priority information of different tasks and schedule the data processing tasks under the guidance of the non-preemptive priority queuing theory.

Experiments have shown that the proposed streaming data priority scheduling mechanism can: 1) effectively distinguish different types of tasks and the task priority differences among different vehicles; 2) achieve reasonable scheduling; and 3) reduce the end-to-end latency of higher-priority tasks by up to 46% than low-priority tasks. Thereby, the proposed mechanism meets the demands of differential service level for AVs.

Acknowledgment

Jie Tang is the corresponding author of this paper. This work is supported by Guangdong R&D Key Project of China under Grant No. 2018B010107003, by Guangdong Natural Science Foundation under Grant No. 2018A030310408, 2021A1515011755.

References

[1] "Autopilot," Tesla, https://www.tesla.com/autopilot.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[3] "Spark Overview," Apache Spark, https://spark.apache.org/docs/latest/ (accessed Feb. 12, 2021).

[4] "Spark Streaming," Databricks, https://databricks.com/glossary/what-is-spark-streaming (accessed Feb. 12, 2021).

[5] Didi Chuxing, GAIA Initiative, Chengdu, China, 2016, Accessed on: Feb. 12, 2021. [Online]. Available: https://gaia.didichuxing.com.

[6] Q. Sun, "Research and optimization of job scheduling algorithm based on Hadoop," M.S. thesis, Xi'an University of Technology, Xi'an, China, 2016, Accessed on: Feb. 12, 2021. [Online]. Available: https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201801&filename=1017853391.nh.

[7] L. Gu, "Research of Hadoop Job Scheduling Based on Priority and Reliability in Cloud Computing Environment," M.S. thesis, Hunan University, Changsha, China, 2013, Accessed on: Feb. 12, 2021. [Online]. Available:https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201402&filename=1014167259.nh.

[8] K. Liu, "Research of Task Scheduling Strategy for Heterogeneous Cluster in Spark Computing Environment," M.S. thesis, Hunan University, Changsha, China, 2018, Accessed on: Feb. 12, 2021. [Online]. Available: https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD201901&filename=1018147704.nh.

[9] J. Tang, M. Xu, S. Fu, and K. Huang. "A scheduling optimization technique based on reuse in spark to defend against apt attack," in *Tsinghua Science and Technology*, vol. 23, no. 5, pp. 550-560, Oct. 2018, doi: https://doi.org/10.26599/TST.2018.9010022.

[10] M. T. Islam, S. Karunasekera, and R. Buyya, "dSpark: Deadline-Based Resource Allocation for Big Data Applications in Apache Spark," in 2017 IEEE 13th International Conference on e-Science (e-Science), Auckland, Oct. 2017, pp. 89-98, doi: 10.1109/eScience.2017.21.

[11] T. Ajila and S. Majumdar, "Data Driven Priority Scheduling on a Spark Streaming System," 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Larnaca, Cyprus, 2019, pp. 561-568, doi: 10.1109/CCGRID.2019.00072.

[12] R. Birke, M. Björkqvist, E. Kalyvianaki, and L. Y. Chen, "Meeting Latency Target in Transient Burst: A Case on Spark Streaming," 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, 2017, pp. 149-158, doi: 10.1109/IC2E.2017.17.

[13] H. Jin, F. Chen, S. Wu, Y. Yao, Z.Liu, L. Gu, and Y. Zhou, "Towards Low-Latency Batched Stream Processing by Pre-Scheduling," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 710-722, 2019, doi: https://doi.org/10.1109/TPDS.2018.2866581.

[14] D. Cheng, X. Zhou, Y. Wang and C. Jiang, "Adaptive Scheduling Parallel Jobs with Dynamic Batching in Spark Streaming," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2672-2685, 1 Dec. 2018, doi: 10.1109/TPDS.2018.284623.