Speeding up Analysis of Archived Sensor Data with Columnar Layout for Tape

Ken Iizawa*, Seiji Motoe†, Yuji Yazawa† and Masahisa Tamura*
*Platform Innovation Project
Fujitsu Laboratories Limited
Kawasaki, Japan
Email: {iizawa.ken, masahisa}@fujitsu.com
†Connected Advanced Development Division
TOYOTA MOTOR CORPORATION
Chiyoda, Japan
Email: {seiji_motoe, yuji_yazawa}@mail.toyota.co.jp

***Abstract*—Analyzing archived stream data such as sensor data, packet data, and log data provides valuable insights into past events. Tape technology has been improving in both capacity and performance and thus is suitable for archiving such a large amount of stream data at low cost. However, due to tape's performance characteristics, read performance is poor when data is stored on tape in the same format as on SSD or HDD. In this paper, we propose a method to improve read performance by placing data on tape in a columnar layout aware of physical structure of tape called wrap. Our preliminary evaluation using a realistic workload shows our method to be 53% faster than traditional wrap-unaware columnar layout.**

***Index Terms*—tape, connected car, sensor data**

I. Introduction

Real-time monitoring of stream data such as sensor data, packet data, and log data is widely used for monitoring the status or detecting failures of the device that generated the stream data. In addition, analyzing archived stream data provides valuable insights into past events. For example, analysis of archived packet data can be used for forensic purposes in the investigation of cyber-attacks.

Due to the prevalence of IoT, an increasing number of IoT devices are generating a large amount of structured stream data. For example, a connected car periodically generates data called CAN bus data consisting of a timestamp and a large number of sensor values. Analysis of archived CAN bus data provides various insights. Such insights include identifying a driver [1], analyzing a driver's behavior [2] [3], diagnosing a car [4], improving fuel efficiency [5], and detecting a lane-change [6].

Long-term archiving of CAN bus data is challenging due to its huge volume. The volume of CAN bus data generated by a single connected car is large, and the number of connected cars is increasing rapidly. It is estimated that tens of millions of connected cars will be generating hundreds of megabytes of CAN bus data per month in 2025 [7]. Given such a large amount of CAN bus data, it is important to archive CAN bus data at the lowest possible cost.

Tape is suitable for long-term archiving of a large amount of data. Tape is the cheapest of all storage devices and this trend is expected to continue [8]. In addition, tape technology has also been improving in sequential access performance as a result of the increase in recording density.

However, while tape's sequential access performance has been improving, its random access performance remains poor. As we show in Section II, when sensor data is placed on tape in the same format as on SSD or HDD, analytical workloads such as data mining and machine learning access the data in a non-sequential manner.

A workaround is to cache frequently accessed data in other fast storage devices such as SSD or HDD. However, analytical workloads are known to often perform the same processing on different portions of existing data [9], accessing a large portion of data only a few times. In such workloads, caching data in SSD or HDD is not cost-effective.

In this paper, we propose a method to improve read performance by placing data on tape in a columnar layout aware of physical structure of tape called wrap.

Our preliminary evaluation using a realistic workload shows our method to be 53% faster than traditional wrap-unaware columnar layout.

The remaining of this paper is organized as follows. In Section II, we give a background on tape technology and clarify the problems that analytical workloads encounter when reading data on tape. In Section III, we propose Wrap-aware Columnar Layout. In Section IV, we propose an architecture to store data in Wrap-aware Columnar Layout. The results of our preliminary evaluation using a realistic workload are shown in Section V. We present related work in Section VI. Finally, Section VII concludes the paper.

II. Background

A. Tape performance characteristics

Linear Tape-Open (LTO) is a widely used magnetic tape technology. Figure 1 shows the head movement when reading data on a tape cartridge. BOT and EOT stand for beginning of tape and end of tape respectively.

A tape cartridge consists of multiple recording areas called wraps. In the case of LTO-7, the number of wraps is 112.

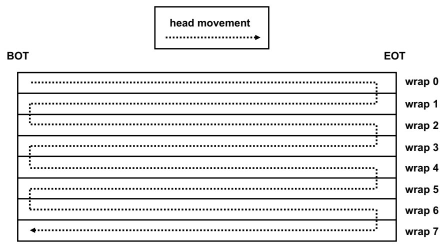


Fig. 1. Head movement when reading data on tape cartridge

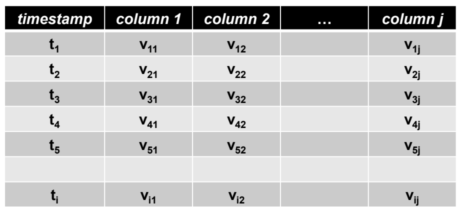
When reading data from the entire tape cartridge, the head starts reading from BOT toward EOT of the first wrap. When the head reaches EOT of the wrap, it switches to the next wrap and continues reading while moving toward BOT.

Tapes have the following performance characteristics.

- Sequential access performance is high.
- Random reads are slow because they issue costly seeks between reads.

B. Challenges of storing data in column-oriented format on tape

As an example, we assume stream data shown in Figure 2.



timestamp	*column 1*	*column 2*	...	*column j*
t_1	v_{11}	v_{12}		v_{1j}
t_2	v_{21}	v_{22}		v_{2j}
t_3	v_{31}	v_{32}		v_{3j}
t_4	v_{41}	v_{42}		v_{4j}
t_5	v_{51}	v_{52}		v_{5j}
t_i	v_{i1}	v_{i2}		v_{ij}

Fig. 2. Example of stream data

Each record consists of a timestamp and many columns. One such example is CAN bus data generated by connected cars.

File formats for storing stream data are classified into row-oriented format and column-oriented format. Row-oriented format, such as Avro, stores records by row as shown in Figure 3. Column-oriented format, such as Parquet, stores records by column as shown in Figure 4.

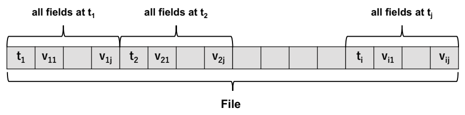


Fig. 3. Row-oriented format

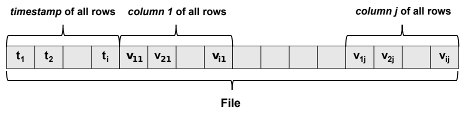


Fig. 4. Column-oriented format

Analytical workloads such as sensor data analysis read a few columns of many rows instead of reading all the columns of a few rows [10]. Therefore, column-oriented format is suitable for such workloads, in which each column can be read sequentially.

However, storing stream data in column-oriented format on tape involves the following problems.

Structured stream data arrives at storage systems row by row. Thus, in order to store it in column-oriented format, the data first needs to be buffered and then converted into column-oriented format.

The ideal data layout for analytical workloads is one in which a single column-oriented format file is stored in one cartridge. An example is shown in Figure 5. In this case, since all the data in a column are continuously arranged, the column can be read sequentially, thus making the best use of tape's sequential access performance.

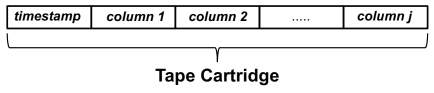


Fig. 5. One column-oriented file stored in one cartridge

However, in order to realize the ideal data layout, stream data needs to be buffered until its size reaches the size of an entire cartridge and grouped by columns and then written to the cartridge as a single file. Since this grouping process issues a large number of random reads, the buffer needs to be an expensive and high-performance storage device such as DRAM or SSD. LTO-12, the newest standard on LTO roadmap, has a capacity of 192 TB per cartridge. Moreover, tapes are expected to continue to increase in capacity in the future. Having a buffer of such a large capacity is costly.

A more feasible way is to use a smaller-capacity buffer, sort the data when the buffer gets full, and write it to the cartridge. In this case, many column-oriented files of the same size as the buffer are stored in a single cartridge (Fig.6). In order to read a column from all files stored in the cartridge, seeks and reads need to be repeated (*cartridge size*)/(*buffer size*) times. This is costly and thus storing data in many column-oriented files does not bring any performance improvement.

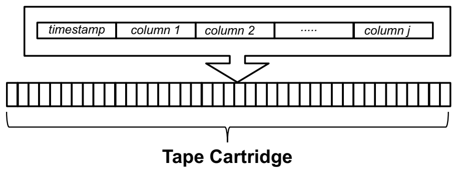


Fig. 6. Many column-oriented files in one cartridge

To solve these problems, in Section III, we propose Wrap-aware Columnar Layout that can improve read performance with a buffer of much smaller size than a tape cartridge.

III. Wrap-aware Columnar Layout

As described in Section II, a tape cartridge has physical structures called wraps. Physical address within a wrap is called LPOS. In the region where data is stored, LPOS takes a value between 2650 and 171144. The left end of Figure 7 corresponds to 2650 and the right end corresponds to 171144. LPOS is a value determined by servo information written in servo bands of a cartridge.

Head movements across wraps can be used for shortcuts. For example, when the head moves from point A on wrap 1 to point B on wrap 2, the head does not need to move to the end of wrap 1. Instead, it can move the shortest distance between the two points (Figure 7).

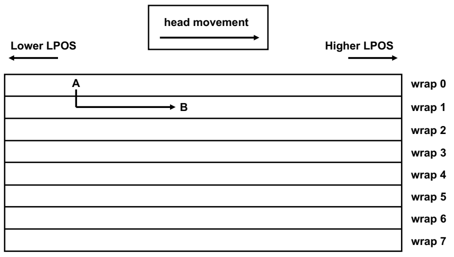


Fig. 7. Head movement when moving between two points on different wraps

Thus, by arranging data in a column across wraps so that its LPOSes on different wraps are close to each other, it is possible to speed up reads to the same column.

We propose to write stream data to a cartridge in the following procedure.

1) When the size of data in the buffer reaches the size of a wrap, the data is grouped by columns, and written to the cartridge as one column-oriented file.
2) Padding is performed so that the next write starts exactly at the beginning of the next wrap.
3) When the size of data in the buffer reaches the size of a wrap again, the data is grouped by columns, and written to the cartridge as one column-oriented file. This time, the arrangement order of columns is reverse to that of the write to the previous wrap.
4) Padding is performed so that the next write starts exactly at the beginning of the next wrap.
5) 1-4 are repeated until the cartridge becomes full.

This method can arrange the data of the same column in close LPOSes on different wraps while keeping the size of the buffer small. As a result, as shown in Figure 8, when reading a column across wraps, the data of the next wrap can be read by moving the head between wraps just after reading the data of the previous wrap.

However, it should be noted that Figure 8 shows the ideal case. Actually, it is almost impossible to arrange a column in exactly the same LPOSes due to factors described in Section IV. Thus, some seeks occur after head movements between wraps. However, even in such a case, shortening seek distance improves read performance.

IV. Architecture

We show an architecture to archive stream data in Wrap-aware Columnar Layout in Figure 9.

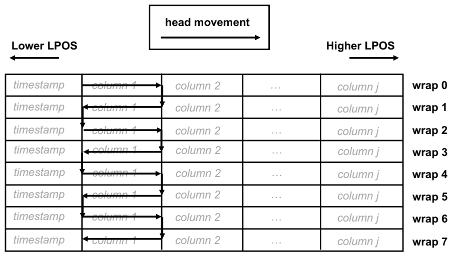


Fig. 8. Head movement when reading the same column from multiple wraps

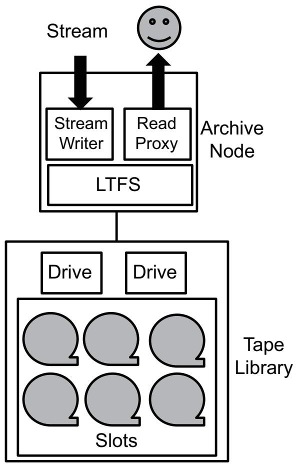


Fig. 9. Architecture

Archive Node is connected to Tape Library. Stream Writer and Read Proxy run on Archive Node.

Stream Writer buffers and converts stream data and then writes it to a cartridge in Wrap-aware Columnar Layout.

Read Proxy reads and returns data from the cartridge in response to a read request from clients. We assume clients to be analytical applications such as Apache Spark [11].

Cartridges are formatted with LTFS [12], which is a POSIX -compliant filesystem for tape.

Next, we describe the details of Stream Writer and Read Proxy.

A. Stream Writer

When the number of records of the stream data on the buffer reaches the predetermined threshold, Stream Writer converts the data into a column-oriented format file and writes it to the cartridge. Since we assume clients to be analytical applications such as Apache Spark, we use Parquet, which is widely used in Hadoop ecosystem, as column-oriented format.

In order for Stream Writer to be able to write a column-oriented format file of the same size as a wrap, it needs to know the following two pieces of information.

First, it needs to know the capacity of a wrap to write to. Since LTO specifications do not define the size of a wrap, we have to determine the size of wraps through measurements. We performed the following measurements before conducting experiments described in Section V.

In the measurements, We wrote many 100 MB files to a cartridge. After writing each file, we synced the tape drive's buffer to ensure that the file is persisted on tape. Then, we checked the LPOS of the current head position using a SCSI command called REQUEST SENSE. The size of a wrap was determined to be the total sum of files' sizes that were written to the wrap.

As a result, we determined the size of all wraps of an LTO-7 cartridge to be 54,000 MB. We confirmed that when files of this size are written to wraps, the LPOS at the left end of the files falls within the range of 2650-4500 and the LPOS at the right end falls within the range of 169500-171000.

After writing a file to a wrap, Stream Writer pads the cartridge with 50 MB files until the head reaches the starting position of the next wrap. The starting LPOS of an even-number wrap, where the head moves toward EOT, is 3000 and the starting LPOS of an odd-number wrap, where the head moves toward BOT, is 171000.

Second, Stream Writer needs to know the number of records of stream data corresponding to the capacity of a wrap. Column-oriented file formats such as Parquet use various compression algorithms to compress columns in order to improve space efficiency. Therefore, the actual file size can not be predicted until the stream data is converted to a file. However, we can assume that the compression ratio of a column is stable over time. Thus, we can find the number of records corresponding to the capacity of a wrap through preliminary experiments and use it as a threshold in production environments. If the converted file size exceeds the wrap capacity, Stream Writer can reduce the number of records and retry conversion.

B. Read Proxy

Read Proxy reads data from the cartridge for clients. This process is needed to serialize reads from processes running in parallel, which often happens in analytical applications such as Spark.

A Spark application is split into stages composed of many parallel tasks. Early stages read input data from a storage system. When receiving multiple files as input, Spark launches as many map tasks as the files. Each map task takes one file as input and performs processing. Thus, if one map task takes as input a column-oriented format file stored in a wrap, multiple map tasks access each file in parallel. These accesses are performed wrap by wrap rather than column by column as depicted in Figure 8.

To deal with this problem, we implemented Read Proxy that serializes accesses from multiple clients to files on LTFS. It is implemented as a user-space file system using FUSE. It works as follows.

1) When a map task accesses a special directory, which is the mount point of the user-space filesystem, Read Proxy blocks all the accesses to the directory.
2) Read Proxy reads the files from the cartridge column by column.
3) When Read Proxy reads all the requested columns into memory, it generates one file on `tmpfs`, which is Linux's in-memory file system, for each file stored in the cartridge. The generated files are also in Parquet format and sparse files that contain only the requested columns and the footer extracted from the corresponding source file.
4) Read Proxy returns the corresponding file to each map task.

V. Evaluation

We conducted a preliminary evaluation using a realistic workload to figure out how much read performance improvement Wrap-aware Columnar Layout achieves compared to traditional wrap-unaware columnar layout.

A. Dataset and Workload

We generated a dataset for evaluation in the following procedure. First, using a vehicle simulator, we collected CAN bus data generated by a connected car in one minute. The rate at which records are generated is one record per 100 milliseconds. The number of columns of this CAN data is 966 including a timestamp. Then, we augmented the generated data by repeatedly copying it with different VIN (vehicle identification number) and timestamp. As a result, we acquired a dataset that corresponds to CAN bus data generated by 1,000 connected cars in 6,204 minutes. Thus, the total number of records is 3,722,400,000. The generated records are ordered by timestamp in ascending order.

The dataset was split into Parquet files using one of the following three data layouts. For each data layout, the dataset was stored in a separate cartridge.

- Split the dataset into 1 GB Parquet files (Many Files Layout), corresponding to Figure 6.
- Split the dataset into Parquet files of wrap size and store each of them on each wrap (Wrap-aware Columnar Layout), corresponding to Figure 8.
- Store the entire dataset as a single Parquet file (Single File Layout), corresponding to Figure 5.

As a realistic workload, we assume a Spark application that identifies geographical locations where the car speed is likely to increase. The application reads six out of 966 columns. The information on the six columns is shown in Table I.

Each column is compressed with a different algorithm depending on the column type. Therefore, the size of each column varies greatly depending on its selectivity and the compression algorithm. In addition, timestamp differs greatly in total size across data layouts, which is seemingly due to the characteristics of the encoding used.

The dataset sizes in each layout are 305 GB (Many Files Layout), 324 GB (Wrap-aware Columnar Layout), and 328 GB (Single File Layout). The number of records in this dataset was chosen so that it occupies six wraps on an LTO-7 cartridge when stored in Wrap-aware Columnar Layout.

For the above three data layouts, we measured read time.

TABLE I
COLUMN INFORMATION

column name	column index	column type	total size [MB] (Many Files Layout)	total size [MB] (Wrap-aware Columnar Layout)	total size [MB] (Single File Layout)
VIN	2	string	27	25	24
timestamp	8	long	6,076	25,084	28,998
latitude	9	float	707	707	707
longitude	10	float	335	335	335
speed	49	long	3,723	3,722	3,722
acceleration	67	float	2,016	2,016	2,016

In order to focus on read time instead of the entire execution time of the application, we measured read time with a simple benchmark without running the Spark application. The benchmark measures time taken to read column(s) from the dataset and copy them into tmpfs without uncompressing them, making a sparse file readable from the Spark application.

In Many Files Layout and Single File Layout, the benchmark reads data file by file.

In Wrap-aware Columnar Layout, the benchmark reads data column by column. When reading six columns, it uses a naive scheduling algorithm that always starts reading a column from the first wrap. Thus, whenever it finishes reading a column, the tape head moves back to the first wrap and starts reading the next column.

Before starting all measurements, the following procedure was performed.

1) In order to exclude the seek time to access the footer of a Parquet file from the time to be measured, the footers of all Parquet files are read and pinned in memory. A footer contains data layout information of the entire file and thus needs to be read before columns are read.
2) The tape head is set to the beginning of the first wrap that stores the dataset.
3) Page cache is cleared.

Reading a Parquet footer before reading data degrades read performance severely. Thus, in production environments, we suppose they should be stored separately from data in other storage devices such as HDD.

B. Experimental Setup

The setup consists of a server and a tape library connected by SAS 3.0.

The server consists of two 2.40 GHz 64-bit processors and 192 GB DDR4 RAM. CentOS 7.6 and FUSE 2.9.2 are installed on the server.

The tape library has four LTO-7 drives, only one of which was used for the experiment. Cartridges are formatted with LTFS format specification 2.4.0. Hardware compression of the drive was disabled.

C. Results

Figure 10 shows the results when one column is read. Times labeled ”read” and ”seek” show the actual read and seek times measured by adding traceability to LTFS. Time labeled ”other” shows other overheads.

As expected, in Many Files Layout, total read time is constant regardless of the column to be read, and seek time occupies most of the total read time.

Interestingly, when reading four columns (*latitude*, *longitude*, *speed*, *acceleration*), Wrap-aware Column Layout outperforms Single File Layout. This is because the four columns are placed at lower LPOSes in Wrap-aware Columnar Layout due to their small column indexes (see Table I). On the other hand, the four columns happen to be placed at higher LPOSes (between 140,000 and 90,000) in the first wrap in Single File Layout.

Columns other than *timestamp* have the same size in all layouts, but in Many Files Layout, it takes longer to read columns than in other layouts. This is considered to be due to the overhead involved in switching the head from seek mode to read mode.

Figure 11 shows results when six columns are read.

In Wrap-aware Columnar Layout, seek time is significantly longer than when reading one column. This is partly due to more head movements across wraps caused by the naive scheduling, which moves the tape head back to the first wrap before starting reading the next column. Seek time should be shortened by optimizing scheduling. For example, the scheduler could start reading a column from the last wrap without moving back to the first wrap, or read multiple columns together if they are continuous on a wrap (e.g. *timestamp*, *latitude* and *longitude* in our experiment).

On the other hand, in Single File Layout, seek time is shorter than when reading one column. This is because when reading six columns, the tape head moves forward by reading more columns instead of skipping them.

In all layouts, other overhead is longer when reading six columns than when reading one column. We observed that the overhead is roughly proportional to actual read time. This is considered to be the overhead incurred by FUSE, which LTFS is implemented on and splits a large read request into many small read requests (e.g. 128 KiB).

VI. Related Work

There are many studies that explore ways of improving read performance of tape.

a) Request Scheduling: Reordering read requests can greatly improve read performance.

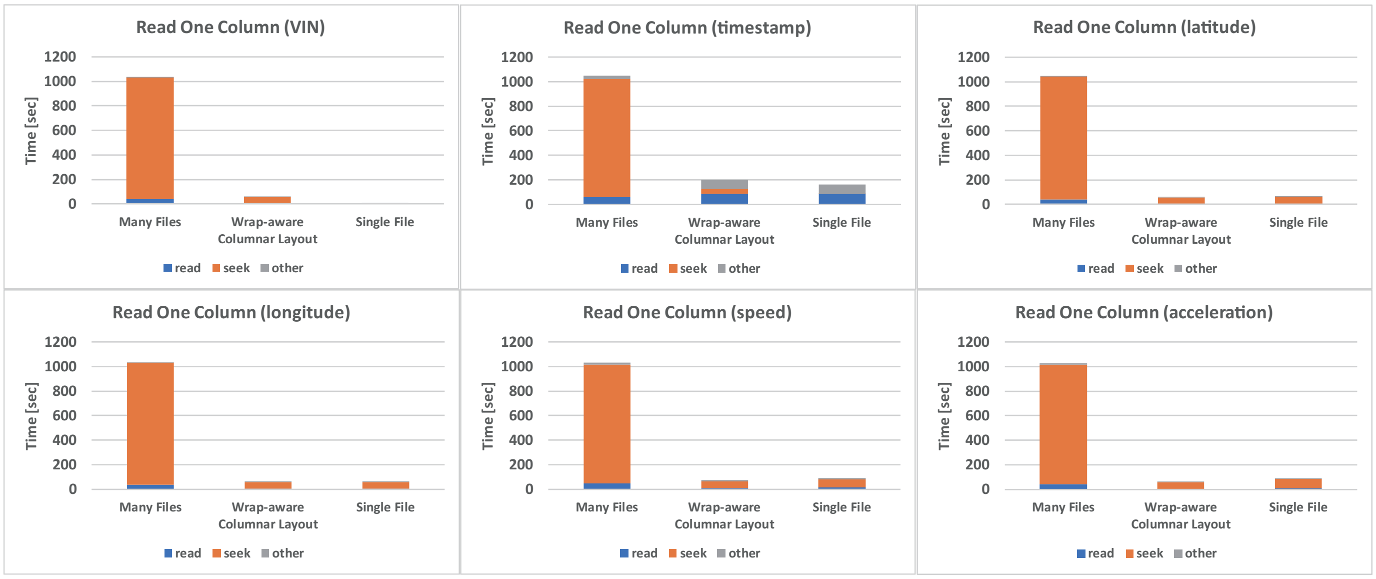


Fig. 10. Read time when reading one column

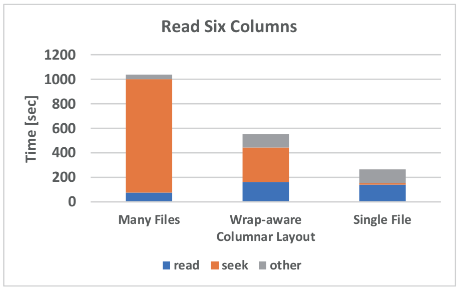


Fig. 11. Read time when reading six columns

There is some work that proposes reordering requests such that they are executed in order of increasing logical addresses [13] [14] [15].

Recently, inspired by other studies [16] [17], Meria [18] showed that reordering requests in consideration of physical locations of accessed data can much further improve read performance.

b) Data Layout Optimization: We are not the first to propose to place data in a wrap-aware manner.

Fujihara et al. [19] proposed to place unstructured stream data such as video from multiple surveillance cameras in a wrap-aware manner so that each stream can be read fast.

However, to the best of our knowledge, ours is the first study to convert structured stream data into a columnar format, place it in a wrap-aware manner and perform quantitative performance analysis using a realistic workload.

c) Reducing Media Exchange: Separability of expensive drives from inexpensive cartridges allows users to increase the ratio of cartridges to drives and makes tape technology the cheapest of all storage devices.

On the flip side, the overhead of exchanging cartridges has been a major cause of performance degradation when using tape. Bessone et al. proposed to delay a cartridge exchange until the amount of requested data reaches a threshold to improve read throughput [20].

Kathpal et al. proposed to schedule map tasks of a MapReduce job performed on tape in such a way that data in the same cartridge can be processed together [13].

Iwata et al. [21] proposed to reduce the number of media changes during log analysis by changing the layout between replicas on optical media.

VII. Conclusion

In this paper, we proposed a method to improve read performance by placing data on tape in a columnar layout aware of physical structure of tape called wrap. Our preliminary evaluation using a realistic workload shows our method to be 53% faster than traditional wrap-unaware columnar layout.

We plan to extend this work in the following two directions.

First, we plan to optimize scheduling. Naive scheduling was used in this paper. However, seek time for reading multiple columns can be greatly reduced by changing the order in which the wraps are read to reduce the head movements between wraps, or by reading multiple columns together if they are continuous on the wrap.

Second, we plan to optimize data layout considering the access pattern of columns. One possible optimization is to place frequently accessed columns at LPOSes close to BOT. Another possible optimization is to arrange columns, which tend to be accessed at the same time, continuously on a wrap. With these optimizations, read performance of Wrap-aware Columnar Layout could even outperform that of Single File Layout.

References

[1] T. Wakita, K. Ozawa, C. Miyajima, K. Igarashi, K. Itou, K. Takeda, and F. Itakura, "Driver identification using driving behavior signals," *IEEE*

Conference on Intelligent Transportation Systems, Proceedings, ITSC, vol. 2005, no. 3, pp. 907–912, 2005.

[2] S. Jafarnejad, G. Castignani, and T. Engel, “Towards a real-time driver identification mechanism based on driving sensing data,” *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-March, pp. 1–7, 2018.

[3] U. Fugiglando, E. Massaro, P. Santi, S. Milardo, K. Abida, R. Stahlmann, F. Netter, and C. Ratti, “Driving Behavior Analysis through CAN Bus Data in an Uncontrolled Environment,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 2, pp. 737–748, 2019.

[4] A. Luckow, K. Kennedy, F. Manhardt, E. Djerekarov, B. Vorster, and A. Apon, “Automotive big data: Applications, workloads and infrastructures,” *BigData*, pp. 1201–1210, 2015.

[5] R. Coppola, M. Morisio, and P. Torino, “Connected Car : Technologies , Issues , Future Trends,” *ACM Computing Surveys*, vol. 49, no. 3, pp. 1–36, 2016.

[6] Y. Zheng and J. H. L. Hansen, “Lane-Change Detection From Steering Signal Using Spectral Segmentation and Learning-Based Classification,” *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 1, pp. 14–24, 2017.

[7] S. Taniguchi, “Envisioning Smart Mobility Society in the Connected Future,” *ITU Workshop on the Future of Vehicular Multimedia*, 2019.

[8] E. Eleftheriou, R. Haas, J. Jelitto, M. Lantz, and H. Pozidis, “Trends in storage technologies,” *Data Engineering*, pp. 1–10.

[9] A. D. Popescu, V. Ercegovac, A. Balmin, M. Branco, and A. Ailamaki, “Same queries, different data: Can we predict runtime performance?” *ICDEW*, pp. 275–280, 2012.

[10] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, and S. Madden, “The design and implementation of modern column-oriented database systems,” *Foundations and Trends in Databases*, vol. 5, no. 3, pp. 197–280, 2012.

[11] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling,” *EuroSys*, pp. 265–278, 2010.

[12] D. Pease, A. Amir, L. V. Real, B. Biskeborn, and M. R. Almaden, “The Linear Tape File System,” *MSST*, vol. 4, pp. 1–9, 2010.

[13] A. Kathpal and G. A. N. Yasa, “Nakshatra: Towards running batch analytics on an archive,” *MASCOTS*, vol. 2015-Febru, no. February, pp. 479–482, 2015.

[14] I. Koltsidas, S. Sarafijanovic, M. Petermann, N. Haustein, H. Seipp, R. Haas, J. Jelitto, T. Weigold, E. Childers, D. Pease, and E. Eleftheriou, “Seamlessly integrating disk and tape in a multi-tiered distributed file system,” *ICDE*, vol. 2015-May, pp. 1328–1339, 2015.

[15] J. Schaeffer and A. G. Casanova, “TReqS: The tape REQuest scheduler,” *Journal of Physics: Conference Series*, vol. 331, no. PART 4, 2011.

[16] O. Sandsta and R. Midtstraum, “Low-cost access time model for serpentine tape drives,” *MSST*, pp. 116–126, 1999.

[17] ——, “Improving the Access Time Performance of Serpentine Tape Drives,” *ICDE*, 1999.

[18] G. C. Meliá, “LTO performance,” *HEPiX*, 2018.

[19] S. Fujihara and Y. Oishi, “Writing Multiple Files Simultaneously to a Tape Media,” U.S. Patent 9 128 617, sep 8, 2015.

[20] N. Bessone, G. Cancio, S. Murray, and G. Taurelli, “Increasing the efficiency of tape-based storage backends,” *Journal of Physics: Conference Series*, vol. 219, no. 1 PART 6, 2010.

[21] S. Iwata and K. Shiozawa, “A Simulation Result of Replicating Data with Another Layout for Reducing Media Exchange of Cold Storage,” *HotStorage*, pp. 2–6, 2016.