On-Edge Aggregation Strategies over Industrial Data Produced by Autonomous Guided Vehicles

Piotr Grzesik[1][0000−0001−8868−0765], Paweł Benecki[1][0000−0003−4674−5393], Daniel Kostrzewa[1][0000−0003−2781−3709], Bohdan Shubyn[1,2][0000−0002−3051−1544], and Dariusz Mrozek[1][0000−0001−6764−6656]

[1] Silesian University of Technology, Department of Applied Informatics, Gliwice, Poland
[2] Lviv Polytechnic National University, Department of Telecommunications, Lviv, Ukraine

Abstract. Industrial IoT systems, such as those based on Autonomous Guided Vehicles (AGV), often generate a massive volume of data that needs to be processed and sent over to the cloud or private data centers. The presented research proposes and evaluates the approaches to data aggregation that help reduce the volume of readings from AGVs, by taking advantage of the edge computing paradigm. For the purposes of this article, we developed the processing workflow that retrieves data from AGVs, persists it in the local edge database, aggregates it in predefined time windows, and sends it to the cloud for further processing. We proposed two aggregation methods used in the considered workflow. We evaluated the developed workflow with different data sets and ran the experiments that allowed us to highlight the data volume reduction for each tested scenario. The results of the experiments show that solutions based on edge devices such as Jetson Xavier NX and technologies such as TimescaleDB can be successfully used to reduce the volume of data in pipelines that process data from Autonomous Guided Vehicles. Additionally, the use of edge computing paradigms improves the resilience to data loss in cases of network failures in such industrial systems.

Keywords: cloud computing, edge computing, automated guided vehicles, data aggregations, internet of things, TimescaleDB, edge analytics

1 Introduction

In recent years, we've observed rapid growth of adoption of IoT systems for various use cases, such as manufacturing [11][18][23], environmental monitoring [7][13], smart cities applications [8][16], agriculture [12][15], health monitoring [17][19] among others. In most of the mentioned cases, a massive volume of data is generated. That data often needs to be sent over to the cloud or private data centers, usually over the Internet, which can be challenging if the network connection is unstable or offers limited bandwidth. In order to address that problem, new computing paradigms such as fog [21] and edge [22] computing

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

have been introduced. The main goal of these paradigms is to reduce the volume of data that needs to be sent over to the cloud by performing data processing directly on devices that are closer to the source of data. Additionally, it enables such systems to react faster to the changes in the system, even in situations where the Internet connection is slow, unreliable, or even not available most of the time. Such edge devices are often responsible for ingestion of the data, storage, aggregation of selected metrics, and sending the results to the cloud for further processing.

Focusing on manufacturing and production, we see the growing popularity of Autonomous Guided Vehicles (AGVs) [6] that allow to modernize production lines, improve internal factory logistics, ensure better safety of factory workers, which in turn enables more flexible and agile production systems. These vehicles need to process data coming from various sensors such as lidars, cameras, optical encoders in a time-efficient manner, which we believe can be improved with edge computing-based techniques.

This paper aims to evaluate two selected approaches to data aggregation performed directly on low-powered edge devices, such as Jetson Xavier NX, in the context of using them for enhancing the industrial data acquisition pipeline from AGVs. In the article, we focused on techniques that allow reducing the volume of data that will need to be sent over to the cloud for further processing. The paper is organized as follows. In section 2, we review the related works. In section 3, we describe the considered workflow, data models, and approaches to data aggregation. Section 4 contains a description of the testing environment. In section 5, we present the performed experiments along with the results. Finally, section 6 concludes the results of the paper.

2 Related Works

Scientific literature shows a variety of use cases for running data aggregations and processing directly on edge devices. In [9], Luca Greco et al. propose an edge-based analytical pipeline for real-time analysis of wearable sensor data. The authors of the research selected Raspberry Pi as an edge computing device, Apache Cassandra as a database, and Apache Kafka and Apache Flink for data processing software. They conclude that selected software, except Apache Flink, provided suitable performance and that the proposed processing pipeline architecture can be successfully used for anomaly detection in real-time.

Real-time processing with Raspberry Pi edge devices is also presented by Abdelilah Bouslama et al. [5] for medical, sensor data. The authors proposed a medical system dedicated to monitoring patients with reduced mobility or located in isolated areas. For implementing the proposed solution, the authors used Amazon Web Services cloud offering and edge devices running Node-Red applications.

One of the important problems while processing data streams is detecting data patterns directly on edge devices instead of doing all processing in the cloud. Eduard Renart et al. [20] proposed a processing framework dedicated to

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

stream processing of data from smart city environments. The authors compare their solution to a single-cloud deployment of Apache Storm and Apache Kafka and conclude that for considered workflows, their solution can offer up to 78% latency reduction and 56% computation time reduction.

In another research, Fatos Xhafa et al. [25] mention the challenges related to processing IoT data streams in the context of edge computing. They focus on processing data coming from cars and experimentally evaluate an infrastructure based on a Raspberry Pi device using Node-Red. The authors conclude the paper with performance tests of the proposed infrastructure. They highlight that a single Raspberry Pi is not suitable for the selected use cases as it cannot process all incoming data in a time-efficient manner.

However, Raspberry Pi is frequently used as the edge device, which is visible in Hikmat Yar et al. [26]. The authors proposed a smart home automation system based on the Raspberry Pi device, which serves as a central controlling unit, analytical engine, and storage system for data generated by smart home devices. Thanks to processing sensor data directly at the edge, authors managed to reduce bandwidth, computation, and storage costs.

Edge computing is also used in other areas. M. Safdar Munir et al. [14] proposed an intelligent irrigation system based on the edge computing paradigm. The authors take advantage of edge servers to collect data from sensors, validate it, and preprocess before sending it to the cloud service for further evaluation against trained machine learning models. The authors highlight that the use of the edge computing paradigm allows reducing the volume of data sent over and improves the overall speed and efficiency of the whole system.

Zhong Wu and Chuan Zhou in [24] propose a system dedicated to detecting the riding posture of equestrian athletes to propose improvements to it in real-time, using the edge computing paradigm for data processing. After performing packet loss rate, driving, and riding tests, the authors conclude that the proposed solution can be successfully used in practical applications.

In another paper, Hong Zhang et al. [27] described an object tracking system dedicated to smart cities, based on IoT and edge computing paradigm. The major contribution is the proposal of a correlation filter algorithm for lightweight computation tracking that can successfully run on low-power consumption IoT devices such as Raspberry Pi or Xilinx SoC platforms. The proposed solution offers better tracking accuracy and robustness than comparable existing systems.

In [10], Grzesik et al. evaluate the possibility of running metagenomic analysis in real-time in the edge computing environment. The authors evaluate Jetson Xavier NX in multiple power modes, running basecalling and classification workloads. After performance experiments, the authors conclude that Jetson Xavier NX can serve as a portable, energy-efficient device capable of running metagenomics experiments.

The above examples show that there is a lot of interest in performing data processing and aggregations directly on edge devices, taking advantage of the edge computing paradigm. This paper aims to expand knowledge in this area in the context of processing industrial data from AGVs. We show two strategies for

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

reducing the amount of data transferred to the data center for further analysis, relying on low-powered boards such as Jetson Xavier NX [1] and software such as TimescaleDB [3].

3 Analytical workflow, data models, and aggregation methods

The analytical workflow consists of a few steps. Firstly, the AGV client periodically retrieves the data from each AGV and persists it in a local database. A separate process is responsible for running aggregations on data retrieved from the local storage. The results from these aggregations are also persisted separately in a local database. Lastly, the third process is responsible for retrieving the aggregated data, performing optional filtering, and sending the aggregated data to the cloud for further processing. The workflow diagram is presented in Fig. 1.

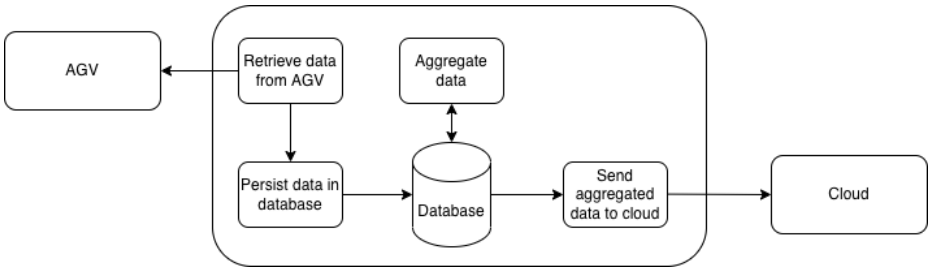

Fig. 1: Diagram of the aggregation workflow on the edge device.

3.1 Data models and aggregation approaches

Each data point collected from the AGV consists of metrics such as battery cell voltage, momentary and cumulative power, energy, and current consumption, cumulative distances, and momentary frequencies. Each reading is additionally timestamped and tagged with the unique AGV identifier. Table 1 shows the structure of a single data point with corresponding data types, their sizes in bytes, and aggregation methods that will be applied to that field. The total size of a single data point from AGV is equal to 84 bytes. The data readings were obtained during real-world experiments with the Formica-1 AGV (Fig. 2) performing different types of workflows and it was collected with the intention of using it for performing predictive maintenance analysis in a cloud-based environment with the use of machine learning algorithms.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

Table 1: Data model of raw readings from AGVs

Value	**Type**	**Size in bytes**	**Aggregation methods**
AGV ID	Integer	4	-
Timestamp	Timestamp	8	-
Momentary current consumption	Decimal	8	AVG, MAX, MIN
Battery cell voltage	Decimal	8	LAST
Momentary power consumption	Decimal	8	AVG, MAX, MIN
Momentary energy consumption	Decimal	8	AVG, MAX, MIN
Cumulative energy consumption	Decimal	8	LAST
Momentary frequency left	Decimal	8	AVG, MAX, MIN
Momentary frequency right	Decimal	8	AVG, MAX, MIN
Cumulative distance left	Decimal	8	LAST
Cumulative distance right	Decimal	8	LAST

Table 2: Data model of aggregated AGV readings

Value	**Type**	**Size in bytes**
AGV ID	Integer	4
Aggregation timestamp	Timestamp	8
Average momentary current	Decimal	8
Max momentary current	Decimal	8
Min momentary current	Decimal	8
Last battery voltage	Decimal	8
Average momentary power	Decimal	8
Min momentary power	Decimal	8
Max momentary power	Decimal	8
Min momentary energy	Decimal	8
Max momentary energy	Decimal	8
Average momentary energy	Decimal	8
Last cumulative energy	Decimal	8
Average momentary frequency left	Decimal	8
Max momentary frequency left	Decimal	8
Min momentary frequency left	Decimal	8
Average momentary frequency right	Decimal	8
Max momentary frequency right	Decimal	8
Min momentary frequency right	Decimal	8
Last cumulative distance left	Decimal	8
Last cumulative distance right	Decimal	8

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

3.1.1 Window aggregation

The first considered method of reducing the volume of the data is called window aggregation and is presented in Algorithm 1. Firstly, the data points collected from AGVs $p \in P$ are filtered against *agvid* for each of N AGVs and the aggregation window start ws and end we, which is shown in line 6 of the algorithm. Afterward, the filtered readings $p' \in P'$ are sorted by the *timestamp* property. Next, each field of the filtered readings $p' \in P'$ is aggregated over time, which is shown in line 8 of the algorithm. For particular types of values, we have decided that we will only report the last recorded value in the aggregation period, while for others, we decided to report average, maximum, and minimal values in the aggregation period. Afterward, the resulting aggregation point is appended to the result array A. After processing data for all AGVs in a given time window, the aggregation window start ws and end we are incremented by time window size Δt (lines 14-15). The processing stops after the algorithm reaches the end aggregation timestamp te (line 3). The structure of aggregated data A, along with the selected aggregation operations for each data point property, is presented in Table 2. The total size of a single aggregation data point is equal to 164 bytes.

Algorithm 1: Data aggregation algorithm

Data: P (all raw readings from AGVs), Δt (aggregation window size), ts (start aggregation timestamp), te (end aggregation timestamp), N (the number of AGVs)
Result: A (all aggregated data points)

```
1  ws ← ts ;                                  /* aggregation window start */
2  we ← ts + Δt ;                             /* aggregation window end */
3  agvid ;                                    /* identifier of a single AGV */
4  while we < te do
5      for agvid ← 1 to N do
6          P' ← ∅ ;                           /* readings set for aggregation window */
7          foreach p ∈ P do
8              if p.agvid = agvid and p.timestamp > ws and
                p.timestamp <= we then
9                  P' ← P' ∪ p;
10             end
11         end
12         Sort set of filtered readings p' ∈ P' by timestamp property;
13         Aggregate each field of filtered readings p' ∈ P' over time;
14         Append aggregation points to result array A;
15     end
16     ws ← ws + Δt;
17     we ← te + Δt;
18 end
```

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

3.1.2 Window aggregation with delta updates

Another method of reducing the volume of the data we considered is called *window aggregation with delta updates* and operates according to Algorithm 2. The aggregation step is the same as in section 3.1.1. However, when sending the aggregation data, an additional step is introduced in which all aggregated readings from AGVs $a \in A$ are first filtered against $agvid$ for each of N AGVs. The result of that operation is assigned to the AF property, which is shown in line 5 of the algorithm. Then, the readings $af \in AF$ are sorted by the timestamp property. In the next step, starting in line 9 of the algorithm, the aggregation results $AF[i]$ are compared with aggregation results for the previous time window $AF[i-1]$ when constructing the data point p. If the value for the field did not change compared to the previous time window, it is not included in the resulting data point p. The data point p is appended to the array of delta updates D with data points that will be sent to the cloud for further processing.

Algorithm 2: Delta updates optimization

Data: A (all aggregated readings from AGVs), N (the number of AGVs)
Result: D (delta updates data points)

```
1  for agvid ← 1 to N do
2      AF ← ∅;
3      foreach a ∈ A do
4          if a.agvid = agvid then
5              AF ← AF ∪ a;
               ;                          /* readings set for given AGV */
6          end
7      end
8      Sort set of filtered readings af ∈ AF by timestamp property and assign
         result to AF;
9      for i ← 0 to len(AF) do
10         if i = 0 then
11             D ← D ∪ AF[i];
12         else
13             Compare each property of AF[i] with AF[i − 1] and include only
                 the fields that are different in the resulting point p;
14             D ← D ∪ p;
15         end
16     end
17 end
```

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

4 Testing environment

The testing environment consists of a group of autonomous guided vehicles equipped with sensors and a central edge device that serves as a database, AGV client, and an analytical engine for the AGV data. Fig. 3 presents the diagram of the described edge computing system. The AGV that was used for data collection was Formica 1, produced by AIUT Ltd. and is presented on Fig.2.

Fig. 2: Autonomous Guided Vehicle Formica 1.

The edge computing device selected for this experiment was Jetson Xavier NX, with its full technical specification presented below [1]:

- CPU - 6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6 MB L2 + 4 MB L3
- GPU - NVIDIA Volta™ architecture with 384 NVIDIA® CUDA® cores and 48 Tensor cores
- Memory - 8 GB 128-bit LPDDR4x 51.2GB/s
- OS Storage - SDHC card (32 GB, class 10)
- DB Storage - Solid State Drive, PNY 500GB M.2 PCIe NVMe XLR8 CS3030
- OS - Ubuntu 18.04.5 LTS with kernel version 4.19.140-tegra

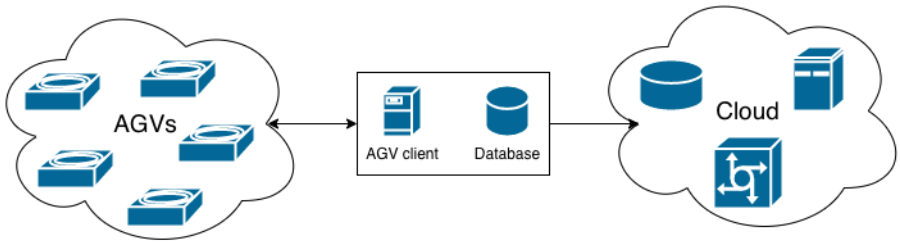

Fig. 3: Diagram of the edge computing system.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

For persistence and analytical aggregations, we selected TimescaleDB, which is an open-source, written in C, time-series database, distributed as an extension to PostgreSQL [2]. It extends traditional tables to form data structures called hypertables, which are abstractions for single, continuous tables. Internally, hypertables are split into chunks, which are implemented using standard PostgreSQL tables[4] and represent data for specific time intervals. It offers support for all PostgreSQL client libraries and SQL operations. It can be used as a drop-in replacement of traditional relational databases that additionally provides significant improvements for storing and processing time-series data, namely, fast data ingestion and support for analytical queries over time windows [3].

5 Performance experiments

To experimentally test the data reduction rates with the aggregation strategies presented in the paper, we simulated the production environment with 10 AGVs generating data based on the actual sensor readings obtained from operational cycles of the Formica 1 vehicle. From each AGV, we read 1,380 data points in regular time intervals. In the experiment, we used three such data sets, with readings generated every 200, 500, and 1,000 milliseconds, which resulted in 41,400 data points used for testing. The generated data was based on actual readings recorded during real-world tests with the Formica-1 AGV.

For the purposes of running the experiments, we prepared several Python software packages developed to carry out each step of the workflow. Firstly, the generated data was persisted to the database to three separate tables, each for different data set based on sampling frequency (200, 500, and 1000 milliseconds between reads). Then, the second package was responsible for generating data aggregates, taking advantage of native database aggregation functions, and the results were persisted to separate tables. Aggregations were run for each data set with different aggregation time windows - 2, 5, and 10 seconds. Lastly, for each of the aggregation approaches, we computed the total number of bytes that would have to be sent to the cloud for further processing and compared the results against the baseline of the number of bytes that would be sent if we forwarded all readings directly to the cloud.

Firstly, we analyzed results obtained for data readings that were collected every 200 milliseconds. With aggregations being done every 2 seconds, the total size of data that needs to be sent to the cloud was reduced from the baseline of 1,159,200 bytes to 226,320 bytes with just aggregations and to 153,680 bytes with additional optimization of sending only data that changed between aggregation results (delta updates). For aggregations done every 5 seconds, we measured 91,840 bytes for the aggregation alone and 73,920 bytes with delta updates, where for the time window of 10 seconds, the results were 45,920 and 40,640 bytes, respectively. The results for that data set are presented in Fig. 4.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

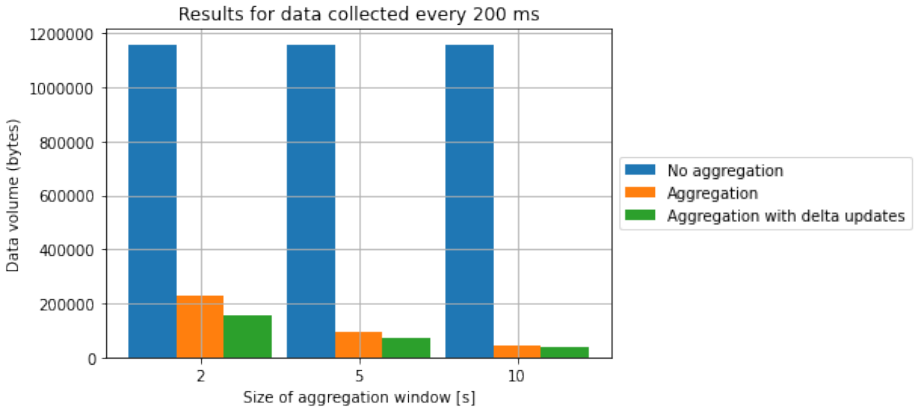

Fig. 4: Dependency between data volume for various data aggregation scenarios and different sizes of the aggregation window for data readings collected every 200 milliseconds.

For data readings collected every 500 milliseconds, we've obtained the following results. The baseline number of bytes was once again 1,159,200 bytes, and it was reduced to 565,800 and 323,960 bytes for a 2-seconds time window, to 226,320 and 153,680 bytes for 5-seconds time window, and finally to 113,160 and 86,040 bytes for 10-seconds aggregation window for aggregation and aggregation with delta updates. The results for that data set are presented in Fig. 5.

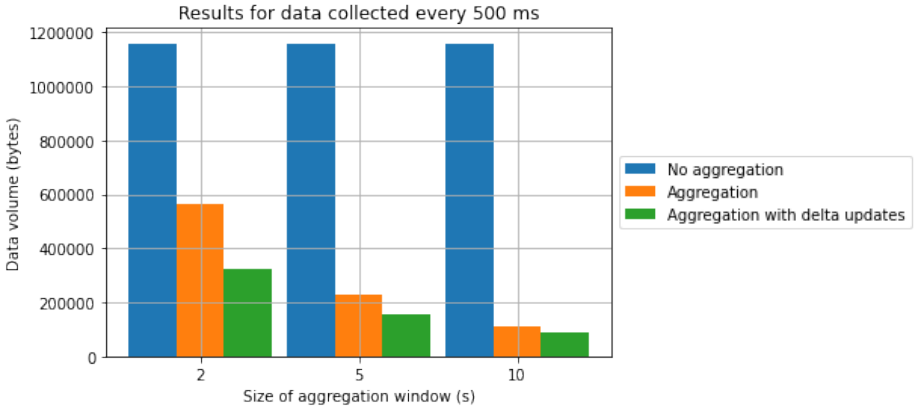

Fig. 5: Dependency between data volume for various data aggregation scenarios and different sizes of the aggregation window for data readings collected every 500 milliseconds.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

Lastly, we have considered the data set that contained readings collected every 1000 milliseconds, which had the same baseline number of 1,159,200 bytes. For the 2-seconds aggregation window, we've achieved a reduction to 1,131,600 and 538,880 bytes, for the 5-seconds window to 452,640 and 275,680 bytes, and for the 10-seconds aggregation window, the results were 226,320 and 153,680 bytes, respectively for aggregation and aggregation with delta updates optimization. The results for that data set are presented in Fig. 6.

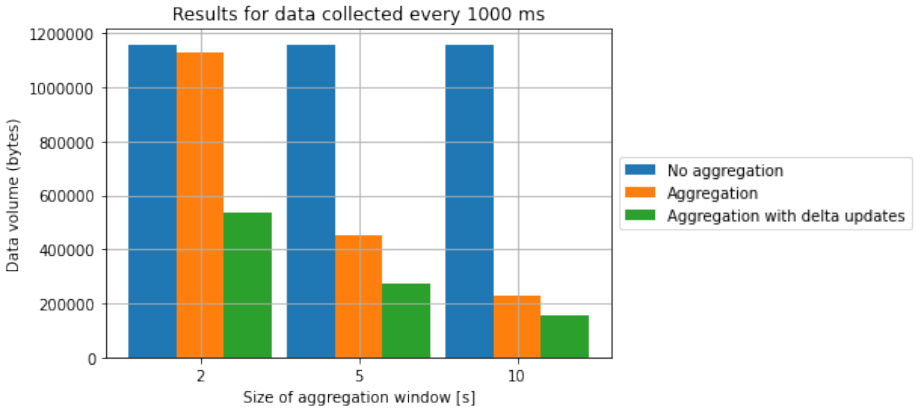

Fig. 6: Dependency between data volume for various data aggregation scenarios and different sizes of the aggregation window for data readings collected every 1000 milliseconds.

Thanks to proposed aggregation techniques, we achieved data volume reduction from 2% for 1000 ms data set and window aggregation method done every 2 seconds to as much as 81.1% for 200 ms data set and window aggregation with delta updates method done every 10 seconds. The data size reductions for each data set and each aggregation method are summarized in Table 3. We also observe that the delta updates technique is more effective for smaller aggregation windows, which is best visible for a 1000 ms data set. Here, it helped to improve data volume reduction from 2% to 44.9% for a 2-seconds aggregation window.

Table 3: Data size reduction for considered aggregation methods

	Aggregation			Agg. with delta updates		
Aggregation time window [s]	2	5	10	2	5	10
Data size reduction for 200 ms data set [%]	67.6	77.3	80.7	72.8	78.6	81.1
Data size reduction for 500 ms data set [%]	43	67.6	75.8	60.5	72.9	77.8
Data size reduction for 1000 ms data set [%]	2	51.2	67.6	44.9	64	72.9

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

6 Concluding Remarks and Future Work

The decision to take advantage of the edge-based data processing pipeline can have multiple benefits for the overall industrial system. As shown in the experiments presented in the paper, both proposed aggregation strategies can greatly help with reducing the volume of readings data from AGVs that need to be send over to the cloud for further processing. In addition to reducing the use of the network bandwidth, using edge storage system based on TimescaleDB database improves resiliency to network outages, because the data is additionally persisted on the edge device, which prevents potential data loss in such scenarios. The aggregated data available at the edge of the network can also be used to make quicker decisions based on the state of the system and improve reaction time to changing environments. In our paper, we proved that the solution based on such edge devices like Jetson Xavier NX and using technologies such as TimescaleDB could successfully help reduce the volume of data in industrial data acquisition pipelines for Autonomous Guided Vehicles. In the future works, we plan to expand the proposed algorithms to evaluate if we can achieve better efficiency and data volume reduction, while preserving data quality, exploring techniques such as GPU acceleration.

Acknowledgments

The research was supported by the Polish Ministry of Science and Higher Education as a part of the CyPhiS program at the Silesian University of Technology, Gliwice, Poland (Contract No. POWR.03.02.00-00-I007/17-00), the Norway Grants 2014-2021 operated by the National Centre for Research and Development under the project "Automated Guided Vehicles integrated with Collaborative Robots for Smart Industry Perspective" (Project Contract no.: NOR/POLNOR/CoBotAGV/0027/2019-00) and by Statutory Research funds of Department of Applied Informatics, Silesian University of Technology, Gliwice, Poland (grant No BK/RAu7/2022).

References

1. Jetson Xavier NX specification (accessed on February 5th, 2022), https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit
2. PostgreSQL documentation (accessed on February 9th, 2022), https://www.postgresql.org/about/
3. TimescaleDB documentation (accessed on February 9th, 2022), https://docs.timescale.com/latest/introduction
4. TimescaleDB : SQL made scalable for time-series data (2017), https://pdfs.semanticscholar.org/049a/af11fa98525b663da18f39d5dcc5d345eb9a.pdf
5. Bouslama, A., Laaziz, Y., Tali, A., Mohamed, E.: Aws and iot for real-time remote medical monitoring. International Journal of Intelligent Enterprise 6, 293 – 310 (04 2019)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

6. Cupek, R., Drewniak, M., Fojcik, M., Kyrkjebø, E., Lin, J., Mrozek, D., Ovsthus, K., Ziebinski, A.: Autonomous Guided Vehicles for Smart Industries – The State-of-the-Art and Research Challenges, pp. 330–343 (06 2020)
7. Fadhel, M., Sekerinski, E., Yao, S.: A Comparison of Time Series Databases for Storing Water Quality Data, pp. 302–313 (04 2019)
8. Gaur, A., Scotney, B., Parr, G., McClean, S.: Smart city architecture and its applications based on iot. Procedia computer science 52, 1089–1094 (2015)
9. Greco, L., Ritrovato, P., Xhafa, F.: An edge-stream computing infrastructure for real-time analysis of wearable sensors data. Future Generation Computer Systems 93, 515–528 (2019), https://www.sciencedirect.com/science/article/pii/S0167739X18314031
10. Grzesik, P., Mrozek, D.: Metagenomic analysis at the edge with jetson xavier nx. In: International Conference on Computational Science. pp. 500–511. Springer (2021)
11. Hu, L., Miao, Y., Wu, G., Hassan, M.M., Humar, I.: irobot-factory: An intelligent robot factory based on cognitive manufacturing and edge computing. Future Generation Computer Systems 90, 569–577 (2019), https://www.sciencedirect.com/science/article/pii/S0167739X1831183X
12. Jaiganesh, S., Gunaseelan, K., Ellappan, V.: Iot agriculture to improve food and farming technology. In: 2017 Conference on Emerging Devices and Smart Systems (ICEDSS). pp. 260–266 (2017)
13. Liu, X., Nielsen, P.: Air quality monitoring system and benchmarking. pp. 459–470 (08 2017)
14. Munir, M.S., Bajwa, I.S., Ashraf, A., Anwar, W., Rashid, R.: Intelligent and smart irrigation system using edge computing and iot. Complexity 2021, 6691571 (Feb 2021), https://doi.org/10.1155/2021/6691571
15. Nandyala, C.S., Kim, H.K.: Green iot agriculture and healthcare application (gaha). International Journal of Smart Home 10(4), 289–300 (2016)
16. Neelakandan, S., Berlin, M., Tripathi, S., Devi, V.B., Bhardwaj, I., Arulkumar, N.: Iot-based traffic prediction and traffic signal control system for smart city. Soft Computing 25(18), 12241–12248 (2021)
17. Paul, A., Pinjari, H., Hong, W.H., Seo, H., Rho, S.: Fog computing-based IoT for health monitoring system. Journal of Sensors 2018, 1–7 (10 2018)
18. Raileanu, S., Borangiu, T., Morariu, O., Iacob, I.: Edge computing in industrial iot framework for cloud-based manufacturing control. In: 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC). pp. 261–266 (2018)
19. Rajavel, R., Ravichandran, S.K., Harimoorthy, K., Nagappan, P., Gobichettipalayam, K.R.: Iot-based smart healthcare video surveillance system using edge computing. Journal of Ambient Intelligence and Humanized Computing (Mar 2021), https://doi.org/10.1007/s12652-021-03157-1
20. Renart, E.G., Diaz-Montes, J., Parashar, M.: Data-driven stream processing at the edge. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC). pp. 31–40 (2017)
21. Sabireen, H., Neelanarayanan, V.: A review on fog computing: architecture, fog with iot, algorithms and research challenges. Ict Express 7(2), 162–176 (2021)
22. Singh, S.: Optimize cloud computations using edge computing. In: 2017 International Conference on Big Data, IoT and Data Science (BID). pp. 49–53 (Dec 2017)
23. Wang, X., Garg, S., Lin, H., Kaddoum, G., Hu, J., Alhamid, M.F.: An intelligent uav based data aggregation algorithm for 5g-enabled internet of things. Computer Networks 185, 107628 (2021), https://www.sciencedirect.com/science/article/pii/S138912862031255X

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39

24. Wu, Z., Zhou, C.: Equestrian sports posture information detection and information service resource aggregation system based on mobile edge computing. Mobile Information Systems 2021, 4741912 (Jul 2021), https://doi.org/10.1155/2021/4741912
25. Xhafa, F., Kilic, B., Krause, P.: Evaluation of iot stream processing at edge computing layer for semantic data enrichment. Future Generation Computer Systems 105, 730–736 (2020), https://www.sciencedirect.com/science/article/pii/S0167739X19321296
26. Yar, H., Imran, A.S., Khan, Z.A., Sajjad, M., Kastrati, Z.: Towards smart home automation using iot-enabled edge-computing paradigm. Sensors 21(14) (2021), https://www.mdpi.com/1424-8220/21/14/4932
27. Zhang, H., Zhang, Z., Zhang, L., Yang, Y., Kang, Q., Sun, D.: Object tracking for a smart city using iot and edge computing. Sensors 19(9) (2019), https://www.mdpi.com/1424-8220/19/9/1987

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_39