Scalable Model-Based Management of Massive High Frequency Wind Turbine Data with ModelarDB

Abduvakhobov, Abduvoris; Jensen, Søren Kejser; Pedersen, Torben Bach; Thomsen, Christian

Published in:
Proceedings of the VLDB Endowment

DOI (link to publication from Publisher):
10.14778/3704965.3704978

Creative Commons License
CC BY-NC-ND 4.0

Publication date:
2024

Document Version
Other version

Link to publication from Aalborg University

Citation for published version (APA):
Abduvakhobov, A., Jensen, S. K., Pedersen, T. B., & Thomsen, C. (2024). Scalable Model-Based Management of Massive High Frequency Wind Turbine Data with ModelarDB. *Proceedings of the VLDB Endowment*, *17*(13), 4723-4732. https://doi.org/10.14778/3704965.3704978

Scalable Model-Based Management of Massive High Frequency Wind Turbine Data with ModelarDB [Scalable Data Science]

Abduvoris Abduvakhobov
Aalborg University
Aalborg, Denmark
abduvorisa@cs.aau.dk

Søren Kejser Jensen
Aalborg University
Aalborg, Denmark
skj@cs.aau.dk

Torben Bach Pedersen
Aalborg University
Aalborg, Denmark
tbp@cs.aau.dk

Christian Thomsen
Aalborg University
Aalborg, Denmark
chr@cs.aau.dk

ABSTRACT

Modern wind turbines are monitored by sensors that generate massive amounts of high frequency time series that are ingested on the edge and then transferred to the cloud where they are stored and analyzed. This results in at least four challenges: (1) Limited hardware makes efficient ingestion necessary to keep up; (2) Limited bandwidth makes data compression necessary; (3) High storage costs as all data must be stored; and (4) Low data quality due to lossy compression methods without error bounds. Practitioners currently use solutions that only solve some of these. In this paper, we evaluate the Time Series Management System ModelarDB, a solution that meets all four challenges by efficiently managing time series across the entire pipeline. We compare it to three commonly used alternatives and evaluate different aspects of them in a realistic edge-to-cloud scenario with real-world datasets. For lossless compression, ModelarDB achieves up to 2x better compression and 1.2x better transfer efficiency. For lossy compression, ModelarDB achieves up to 4.6x better compression and 10x better transfer efficiency, or similar compression with orders of magnitude less error.

PVLDB Reference Format:
Abduvoris Abduvakhobov, Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Scalable Model-Based Management of Massive High Frequency Wind Turbine Data with ModelarDB [Scalable Data Science]. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/aabduvakhobov/ModelarDB-Analyzer.

1 INTRODUCTION

To maintain wind turbines and optimize energy production, wind turbine manufacturers and owners use data analytics, e.g., OLAP queries and complex data mining such as forecasting and anomaly detection as shown in Figure 1. Thus, wind turbines are monitored by many high-quality sensors that generate vast amounts of time series at high frequencies. For example, a wind turbine generating 2500 time series sampled at 100 Hz produces over 321 GiB of data per day assuming timestamps and values use 8 bytes each [29]. In our decade-long experience with Renewable Energy Sources (RES) data, we had access to real-life wind turbine datasets with Sampling Intervals (SIs) typically ranging from 10ms to 2s. According to our industry partners, practitioners collect data on the edge and transfer it to the cloud for analytics. The edge nodes' hardware is similar to low-end commodity PCs. They collect data and help manage wind turbines. They use a ring buffer, so old data is deleted when new data arrives. Thus, the old data must be transferred to the cloud for permanent storage. This is done using connections with very limited bandwidth as they are shared between wind turbines, i.e., generally 512 Kbit/s to 10 Mbit/s per wind turbine. The data is stored in the cloud for as long as possible due to business requirements and there are no established practices for decaying data. A wind turbine owner told us they would never purchase a wind turbine if all its data was not available. Thus, practitioners face these challenges:

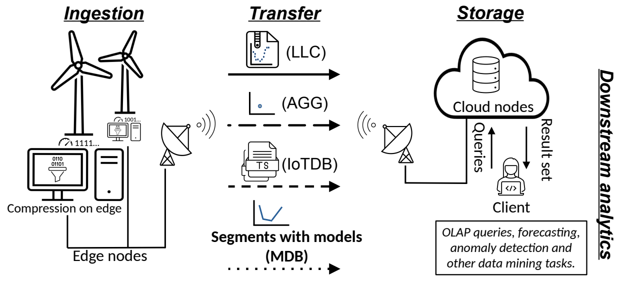


Figure 1: Architecture of RES data management with different lossless and lossy compression methods used on the edge.

Challenge 1: Limited Hardware. Ingestion must keep up with sampling despite limited hardware, e.g., no GPU. Thus, resource intensive methods like autoencoders are not useable [10, 42, 48, 59].

Challenge 2: Limited Bandwidth. The bandwidth between edge and cloud can be as low as 512 Kbit/s so compression is needed.

Challenge 3: High Storage Cost. Storing high frequency time series in the cloud is prohibitively expensive, e.g., one year of data for a wind turbine that generates 321 GiB daily costs ~18,510$ on

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Amazon S3 [3]. All data must be kept for a wind turbine's lifetime (i.e., ~20–25 years) due to business requirements such as warranty.

Challenge 4: Low Data Quality After Compression. To lower bandwidth and storage use, lossy compression without error bounds is often used, but the lack of error bounds can impact analytics [52].

According to our industry partners, practitioners currently use two solutions that only solve a subset of these as shown in Table 1:

Lossless Compression through Big Data Formats (LLC). The edge nodes store time series in big data file formats with lossless compression like Apache Parquet or Apache ORC and these files are transferred to the cloud. This provides fast ingestion and high data quality, but very low compression factor [39] as shown in Table 1.

Unbounded Lossy Compression through Simple Aggregates (AGG). Like LLC, but the edge nodes store simple aggregates for a static time interval, e.g., 10-minute means. This provides fast ingestion and high compression factor, but low data quality due to the use of lossy compression without an error bound as shown in Table 1.

Although not used by our industry partners, we include the state-of-the-art Time Series Management System (TSMS) [30, 31] *Apache IoTDB (IoTDB)* [53, 54] as it supports lossless and lossy compression and data transfer. To optimize data size and quality, the TSMS *ModelarDB (MDB)* [32–35] is developed and evaluated in research projects with practitioners [1, 52]. MDB uses error-bounded *model-based* compression, i.e., using *models* from which the original values can be reconstructed within a user-defined pointwise relative error bound (possibly 0%). MDB's efficiency comes from two key insights about RES data. (1) for some time series it is okay to trade a small amount of error for significantly better compression while for others it is not. Thus, MDB supports both lossless and error-bounded lossy compression. (2) the time series have long subsequences that can be efficiently represented using simple functions, e.g., polynomials, while others cannot. Thus, MDB implements multiple types of models and automatically uses the best one for each subsequence.

MDB can be configured to use different query engines and data stores, e.g., it can be configured to be lightweight on the edge and scalable in the cloud. On the edge, it efficiently compresses data points into *segments* with metadata and models. These segments are transferred to the cloud, thus significantly reducing the amount of bandwidth and storage required. Clients can query MDB using SQL. Thus, MDB provides a complete data management pipeline across edge and cloud. More details about MDB are in Section 3.

In this paper, we evaluate how MDB performs in the environment shown in Figure 1 using real-life datasets provided by our industry partners. We compare it to LLC, AGG, and IoTDB using the following research questions which are based on our long experience with RES data and collaboration with industry partners in the MORE project [1] and with a major wind turbine manufacturer:

	Challenge 1	Challenge 2	Challenge 3	Challenge 4
LLC	✓	✗	✗	✓
AGG	✓	✓	✓	✗
IoTDB	✓	✓	(✓)	✓
MDB	✓	✓	✓	✓

Table 1: Challenges solved by each of the evaluated solutions.

RQ1: *How does a high frequency wind turbine dataset compress with the evaluated solutions?* ***RQ1.1***: *How does MDB compare against the lossless solutions in terms of compression factor?* ***RQ1.2***: *How does MDB compare against the lossy solutions in terms of compression factor and data quality?* ***RQ1.3***: *How does the sampling interval of a high frequency wind turbine dataset affect MDB?*

RQ2: *How is the transfer efficiency of the four solutions?*

RQ3: *How well does MDB preserve the data quality of a high frequency wind turbine dataset?* ***RQ3.1***: *What is the compression error of a high frequency wind turbine dataset when compressed using MDB?* ***RQ3.2***: *What is the impact of MDB's lossy compression on the result of downstream analytics?*

In this paper, we make the following contributions: We realize a part of the vision in [29] by evaluating four solution for the practical problem of efficiently managing high frequency wind turbine data across edge and cloud with a focus on MDB and report key insights.

The rest of the paper is structured as follows. Section 2 contains preliminaries. Section 3 describes MDB. Section 4 describes the evaluation setup, i.e., the solutions, evaluation aspects, evaluation metrics, and hardware used. Section 5 discusses the results and presents insights. Section 6 is related work and Section 7 concludes.

2 PRELIMINARIES

A *time series* is a collection of data points $ts = \langle (t_1, v_1), (t_2, v_2), \dots \rangle$ sorted in ascending order by time. A data point (t_i, v_i) consists of a timestamp t_i and a value $v_i \in \mathbb{R}^n$ for a fixed n. If $n = 1$, the time series is *univariate* and if $n > 1$, the time series is *multivariate*. If the time elapsed between consecutive data points is constant, the time series is *regular* and has the *sampling interval* $SI = t_{i+1} - t_i$.

A *signal* is the univariate time series we get when we for each data point (t_i, v_i) in a time series where $v_i = (v_{i,1}, \dots, v_{i,n}) \in \mathbb{R}^n$ extract $(t_i, v_{i,j})$ for a given j such that we get $\langle (t_1, v_{1,j}), (t_2, v_{2,j}), \dots \rangle$.

Time series compression is the process of encoding a bounded time series $ts = \langle (t_1, v_1), \dots, (t_n, v_n) \rangle$ into another representation c by using a function C such that $c = C(ts, \epsilon)$. For *decompression*, another function $\mathcal{D}$ must exist such that $\mathcal{D}(C(ts, \epsilon))$ gives a time series $ts' = \langle (t_1, v'_1), \dots, (t_n, v'_n) \rangle$ where the *relative pointwise error* $e_i \leq \epsilon$ for $e_i = \frac{v_i - v'_i}{v_i}$ when $v_i \neq 0$ and $e_i = 0$ when $v_i = v'_i = 0$. We call ϵ the *error bound* and when $ts' = ts$, the compression is *lossless*.

3 MODELARDB-BASED SOLUTION

MDB [32, 34, 35] is a TSMS that compresses time series to segments with metadata and models on the edge, transfers the segments to the cloud, and executes SQL queries across edge and cloud. MDB consists of a Java library interfaced with query engines and data stores. For example, MDB can use the lightweight RDBMS H2 as its query engine and data store on the edge, and Apache Spark and Apache Cassandra as its query engine and data store in the cloud.

We use H2 as query engine and data store on the edge, and Apache Spark as query engine and Apache ORC files written to a local file system as data store in the cloud. To measure MDB's compression, we measure the size of its Apache ORC files in the cloud. Apache ORC is used instead of Apache Cassandra for a more direct comparison to LLC and AGG as they also use Apache ORC.

MDB's architecture has three sets of components as shown in Figure 2 with suitable query engines and data stores shown in color.

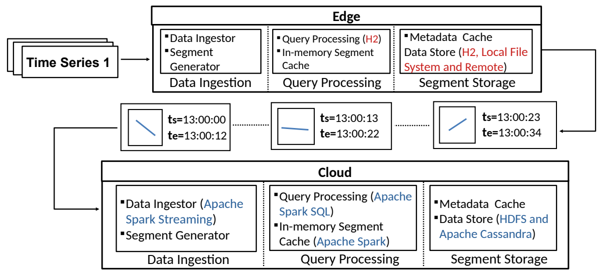


Figure 2: MDB's architecture on the edge and in the cloud.

Data Ingestion. These components ingest data points and compress dynamically sized subsequences to segments using modified versions of *Poor Man's Compression-Mean (PMC)* [39], *Swing Filter (Swing)* [14] and *Gorilla*'s lossless compression method for floating-point values [46]. We call methods that fit models to time series *model types*. PMC returns a constant function, Swing returns a linear function, and Gorilla returns XOR'ed values packed into bytes. These model types are efficient enough to address *Challenge 1 (Limited Hardware)*. MDB ingests data points one at a time and use the first model type to fit a model to them until the error bound is exceeded. Then it switches to the next model type and so on. After evaluating all model types, the model with the best compression factor is stored in a segment with metadata. An example that only uses Swing is shown in Figure 3. Gorilla never exceeds the ϵ as it is lossless. Thus, a user-configurable length bound with a default of 50 is used. Due to its model-based compression and error bound, MDB addresses *Challenge 4 (Low Data Quality After Compression)*.

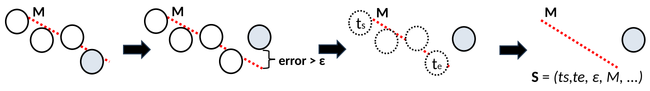


Figure 3: MDB's model-based compression, redrawn [32].

Query Processing. These components execute SQL queries on locally stored segments, e.g., small queries on recent data on the edge and large queries on data from many edge nodes in the cloud. MDB can compute common aggregates directly from segments instead of from reconstructed data points using UDFs and UDAFs.

Segment Storage. These components manage segments in a local data store and transfer them to another MDB instance using Apache Arrow Flight. Transferring and storing time series as highly compressed segments allow MDB to address both *Challenge 2 (Limited Bandwidth)* and *Challenge 3 (High Storage Costs)*, respectively.

4 EVALUATION SETUP

4.1 Baseline Solutions

LLC. Time series written to Apache Parquet or Apache ORC with their default Snappy compression by `Apache Arrow v11.0.0` [16].

AGG. Like LLC, but the mean is computed for different fixed time periods suggested by our industry partners, see Section 4.2.3.

IoTDB. `Apache IoTDB v1.3.1` with its recommended configurations for lossless compression [28]. For lossy compression IoTDB has RLE and TS_2DIFF [26]. Their error is bounded by a pointwise decimal precision limit. We use TS_2DIFF as it provided ~2x better compression than RLE in our experiments. For each dataset, see Section 4.2.1, we use the precision limits for which IoTDB stores values without corruption, e.g., with the active power signal from WTM and precision=7, 377.95465 became -51.54208. For PCD we use precision=1–6 and for MTD and WTM we use precision=1–5.

4.2 Evaluation Aspects

4.2.1 Dataset. Three real-life datasets from our industry partners are used in the evaluation. See Table 2 for a summary. *Power Controller Dataset (PCD)* and *Multiple Turbines Dataset (MTD)* cannot be shared due to NDAs. Our results for *Wind Turbine Measurements (WTM)* [11] can be reproduced as it is a subset of MTD.

PCD. Data from a wind park power controller in a wind park for ~2 years and 4 months. This multivariate time series has SI=150ms, 10 signals, and ~480M data points. The time between consecutive data points can deviate slightly from 150ms, so PCD is made regular through preprocessing as MDB only supports regular series [44].

MTD. Data from wind turbines in several wind parks with a mean period of recording of 11 months. These multivariate time series has SI=2s, 10 signals, and ~258M data points. Four signals are transformations of other in the dataset. These transformed signals are removed as their results very closely match their source signals.

WTM. A subset of MTD published as part of [12]. It is 432,000 data points from one wind turbine collected over 10 days.

Dataset	Length	SI	Signals	Size (ORC)	Period
PCD	~480M	150ms	10	13.3GiB	~28 months
MTD	~258M	2s	6	4.2GiB	~11 months
WTM	432,000	2s	10	10.3MiB	10 days

Table 2: Summary of real-life datasets used in the evaluation.

The datasets mostly contain signals on generated power, wind, and wind turbine configuration, e.g., nacelle direction. Several signals have high periodicity, e.g., the wind signals, and some are correlated with weather forecasts, e.g., generated power. There are signals generated from other, e.g., cosine or sine of signals with degrees and cumulative aggregation of generated power over 1m or 10m intervals. The signals on generated power mostly follow a bimodal distribution. The wind signals follow a normal distribution. There are a few mostly constant signals, e.g., upper and lower power limits which are used to control the generated power. The datasets are used for downstream analytics as discussed in Section 5.3.2.

4.2.2 Error Bound. For MDB, we use these error bounds proposed by our industry partners: 0%, 0.01%, 0.05%, 0.1%, 0.2%, 0.5%, 1%. We also use 5% and 10% to achieve more complete and deeper insights.

4.2.3 Sampling Interval. To measure the impact of different SIs, we downsample the datasets using the SIs and number of data points in Table 3. These SIs were also suggested by our industry partners.

4.2.4 MDB Parameters. MDB can be tuned for wind turbine data using the *length bound* and *batch size* parameters. Length bound is the maximum length of a Gorilla segment. For high frequent wind turbine data, data points often have similar values to adjacent

Dataset	SI	No. Data Points to 1
PCD	1.05 2.1 4.95 10.05 (s); 1 10 (min)	7 14 33 67 400 4000
MTD	6 10 30 (s); 1 10 (min)	3 5 15 300 3000
WTM	6 10 30 (s); 1 10 (min)	3 5 15 300 3000

Table 3: Sampling intervals for aggregation/downsampling.

data points. Thus, PMC and Swing often compress them better than Gorilla. However, short subsequences of data points with irregular values occur and they compress better with Gorilla. Thus, a relatively low length bound of 50 is used so MDB uses Gorilla for these subsequences, but quickly reverts to PMC and Swing. Batch size is the number of segments written to disk or transferred together. Due to the low bandwidth, batch size is set to 1000 to reduce the overhead per segment and amount of data to resend on failure. In Appendix A, the combination of parameters is evaluated and length bound 50 gives the best compression factor in 85% of cases and batch size 1000 gives the best ingestion time in 50% of cases.

4.3 Metrics

To measure the effectiveness of compression methods, we use *Compression Factor (CF)* which is the ratio between the size s of the original dataset and the size s' of the compressed dataset: $CF = s/s'$. The *Transfer Efficiency* is given by the number of data points transferred from the edge node to the cloud node per second. To measure the quality of reconstructed datasets, we use the *Mean Absolute Percentage Error (MAPE)* defined as $MAPE = \frac{1}{n}\sum_{i=1}^{n} e_i$ where n is the number of data points and e_i is the error at the i^{th} data point.

4.4 Infrastructure

Our test infrastructure has an edge node and a cloud node. The edge node has 2 CPU Cores and 4 Threads, 4 GiB RAM, and an HDD. Thus, it is similar to the one in [29]. The cloud node has 16 CPU Cores and 32 Threads, 256 GiB RAM, and 8 SSDs. They both run Ubuntu 20.04 LTS. The edge node's network interface was also limited to 512 Kbit/s and 2.5 Mbit/s to simulate the scenario in [29].

5 FINDINGS AND INSIGHTS

5.1 Compression Effectiveness

In this section, we analyze MDB's compression and compare it to LLC, IoTDB, and AGG. Then, we analyze the impact of SI on MDB.

5.1.1 MDB against the lossless solutions. We first compare LLC and IoTDB to MDB with ϵ=0%, i.e., lossless compression. Figure 4 shows the file sizes of the compressed datasets. Since Apache ORC compresses better than Apache Parquet for all datasets, in line with previous work [32, 34], and they have similar compression times [18, 32, 34], only Apache ORC is used below for LLC. The results show that MDB provides 1.5x, 1.4x and 1.3x better lossless compression than Apache ORC for PCD, MTD and WTM, respectively. MDB also provides 1.2x and 1.1x better compression for PCD and MTD than IoTDB, respectively. Only for the small WTM, IoTDB provides 1.4x better compression than MDB.

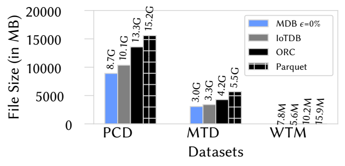


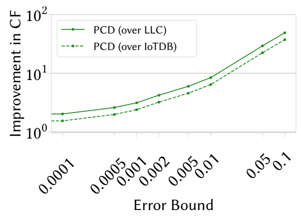
Figure 4: Size of ϵ=0% solutions.

Discussions with our industry partners revealed that while guaranteed data quality is desired, a small pointwise error bound is tolerable. Thus, we compare MDB's error-bounded lossy compression to the lossless solutions. Figures 5a and 5b show MDB's improvement in CF over LLC and IoTDB when $\epsilon > 0$. MDB's improvement in CF over LLC is computed as CF_{MDB}/CF_{LLC} and similarly for IoTDB (and thus they differ by a constant $\frac{CF_{MDB}/CF_{LLC}}{CF_{MDB}/CF_{IoTDB}} = \frac{CF_{IoTDB}}{CF_{LLC}}$). Note that Figure 5a uses a log scale for the y axis. For PCD, ϵ=0.01% leads to more than 2x better compression than LLC. MDB's improvement in CF over LLC for PCD grows to 6x, 8.4x, 29.6x and 48.9x at ϵ=0.5%, ϵ=1%, ϵ=5% and ϵ=10%. In Figure 5b, the improvements in CF for MTD and WTM are much smaller (up to 3.3x for ϵ=10%).

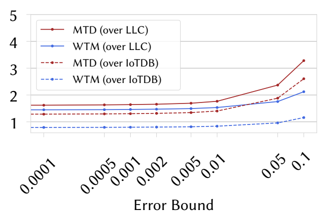
To explain the significant difference in CF between PCD and MTD, we analyze MDB's use of model types for the datasets. The amounts of values compressed by each model type can be seen in Figures 5c and 5d. WTM originates from the same source as MTD and has similar results. Thus, we refer to WTM's results in the extended version B, where we also report the mean and median length of segments. The results for PCD show that as ϵ increases, the use of PMC (Figure 5c) and the mean length of PMC segments increase, while the median length decreases. PMC uses 32 bits and Swing uses up to 128 bits for each segment, while Gorilla uses 1–32 bits for each value. At higher error bounds, PMC constructs few long segments with thousands of values, so the mean length is very long, while most segments are short (up to 20 values) (Appendix C). This explains why PCD, mostly represented by PMC, results in significantly higher CF compared to the other datasets. Measures of dispersion for the datasets also show that there is less variability in PCD values compared to the other datasets and this leads to higher use of PMC for PCD since the values are close to each other.

Gorilla and Swing are more heavily used for MTD and WTM at higher error bounds, which also explains the smaller impact of ϵ on those datasets as segments with those model types require significantly more bits and have shorter mean length than PMC. Figures 5c and 5d also show that even for ϵ=0%, PMC is significantly used for representing PCD (16.2% of the values) and MTD (26.2% of the values), while lossless Gorilla represents the rest. Using a *tiny error bound of 0.01% significantly increases the use of model types for lossy compression* which explains the significant increase in CF between compressing with ϵ=0% and ϵ=0.01%.

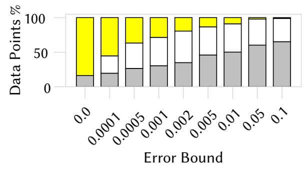
To conclude, we answer **RQ1.1** as follows. For lossless compression, MDB outperforms both LLC and IoTDB. The results showed that *for PCD, allowing a tiny ϵ such as 0.01% can significantly improve* the compression, while at higher error bounds such as 5%, MDB is 29.6x better than LLC. We saw that the dataset with the lowest SI (i.e., PCD) is compressed significantly better by MDB than the other datasets. For PCD, MDB mostly used PMC which had much longer segments than the other model types. For MTD, Swing is more used than PMC for higher error bounds showing that different datasets

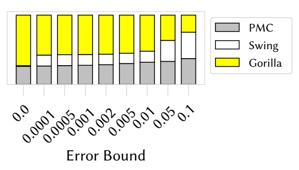


(a) PCD



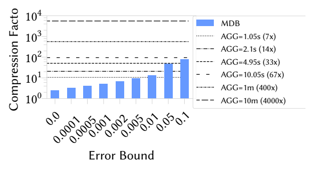
(b) MTD and WTM



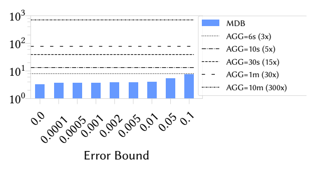


(c) PCD **(d) MTD**

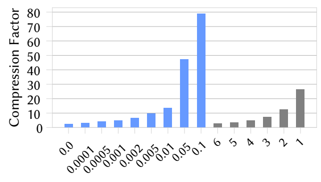
Figure 5: (a-b) Improvement in MDB's CF over LLC and IoTDB for error bounds above 0%. (c-d) Distribution of values compressed by MDB's model types for PCD and MTD.



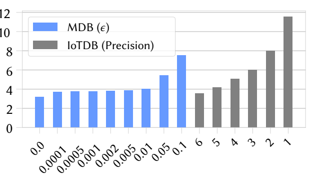
(a) PCD



(b) MTD



(c) PCD



(d) MTD

Figure 6: (a-b) MDB (bars) and AGG (lines) CFs for PCD and MTD. x-axis (i.e., error bound) is only used for MDB. AGG is independent of the error bound. (c-d) MDB and IoTDB's lossy compression's CFs for PCD and MTD.

and error bounds use different model types for compression. This shows the effectiveness of MDB's multi-model compression.

5.1.2 MDB compared to lossy solutions. Figures 6a and 6b compare the CFs of MDB and AGG for PCD and MTD. The CFs for MDB are shown as bars for different values of ϵ. As AGG's error is independent of ϵ, the CFs for AGG are shown as horizontal lines. Figure 6a shows that MDB with ϵ=1% compresses better than aggregating by 7 values (i.e., 7x aggregation) for PCD. MDB at ϵ=5% and ϵ=10% provides comparable CFs to 33x and 67x aggregation, respectively. Compression of MTD (Figure 6b) and WTM (Appendix B) with MDB at ϵ=10% provides comparable CFs to 3x aggregation. Figures 6c-6d show that IoTDB with precision=6 provides comparable compression to MDB at ϵ=0% (i.e., lossless compression) for both PCD and MTD. For PCD, IoTDB with precision=1 achieves a CF

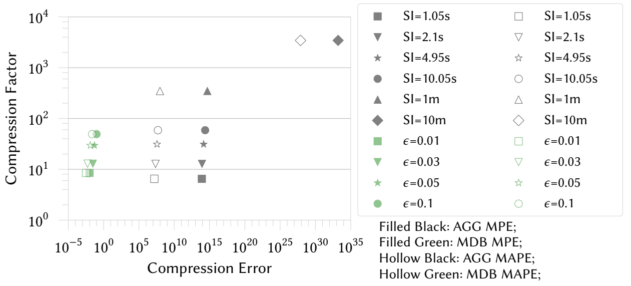


Figure 7: CFs and resulting errors for MDB and AGG on PCD.

that is in between what MDB achieves for ϵ=1% and ϵ=5%. For MTD, we can see that IoTDB with precision=2 achieves CF=8 that is comparable to MDB at ϵ=10%, while with precision=1 it compresses 1.5x better.

In Figure 7, we compare MDB and AGG's MAPE and maximum pointwise error (MPE) for PCD. MAPE for AGG is *3–8 orders of magnitude higher* than for MDB, while the MPE is *9–17 orders of magnitude higher*. As an example, the MPE of AGG with SI=10.05s is equal to ~*two hundred trillion*, while MDB with ϵ=10% provides the same CF and guarantees that the MPE is no more than 10%.

We also compressed the datasets with MDB with $\epsilon > 10\%$ to match AGG's CF with very high SIs. The results show that MDB with ϵ>10% introduces a much smaller error than AGG with the same CF for MTD (Appendix B). Also for MTD and WTM AGG leads to extremely high errors (Appendix B). In addition to very high error, AGG generates *undefined errors* when $v_i = 0$ due to division by zero. We excluded these values when computing errors. In contrast, to maintain the pointwise ϵ, MDB stores $v_i = 0$ without any error.

IoTDB's lossy compression provides low MAPEs. Values close to zero are trivially represented by 0. This results in a MPE of 1 for all datasets and precision limits, which is 10x higher than MDB at ϵ=10%. IoTDB's lossy compression can in some cases result in very large query errors due to rounding of decimal points, see Section 5.3.3. For PCD, IoTDB's MAPEs with precisions 1 and 6 match MDB's MAPEs at ϵ=10% and ϵ=1%, respectively. However, we saw that MDB with ϵ=10% and ϵ=1% provides 3x and 4.6x better compression than IoTDB's lossy compression with precision 1 and 6, respectively. For MTD, IoTDB's MAPE at precision=5 matches MDB at ϵ=0.01% and with precision=1 IoTDB provides similar MAPE to MDB at ϵ=5%. IoTDB's MAPE results for WTM are similar to MTD.

We answer **RQ1.2** as follows. MDB provides comparable CF to AGG, however, in contrast to MDB, AGG *does not provide any data quality guarantees* leading to significant reduction in the quality of compressed values. Unlike MDB's pointwise relative error-bound, AGG removes informative outliers and fluctuations in a dataset that are often critical to certain analytical tasks [32, 34]. Compared to IoTDB's lossy compression, MDB provides *higher CF and similar data quality*. Compared to AGG, MDB achieves as good a compression factor as AGG, but with errors that are *many orders of magnitude smaller, making MDB the far better* solution.

5.1.3 Impact of SI on MDB. To evaluate the impact of SI on MDB, we downsample the datasets as described in Section 4.2.3 and compress them in MDB using different ϵ values. We also compress the downsampled datasets with LLC for comparison. Figure 8 shows how the CFs change for PCD and MTD as we increase SI. As SI increases, MDB's CFs decrease for both datasets showing a negative correlation between MDB's CF and the SI. As PMC and Swing exploit constant and linear patterns, high frequency datasets and higher ϵ, where the difference between two consecutive values tends to be smaller, create more opportunities for storing many values in one segment. For PCD (Figure 8a) with ϵ=0%, the impact of the SI on CF is less significant compared to ϵ=10% where the CF decreases from 79.1 to 7.8 and 2.3 when the SI is increased from 150ms to 1m and 10m, respectively. The CF for PCD, which has the lowest SI (150ms), is affected most as we increase both ϵ and SI. Analyzing all three datasets' measures of dispersion revealed that increasing SI increases variability among the values which makes MDB's compression less effective. However, we can see that MDB at all error bounds compresses better than LLC for all SIs with the only exception being the extreme case of MDB at ϵ=0% for MTD with SI=10m, where the data volumes are small. As an example, for PCD with SI=10m MDB at ϵ=0% still compresses 1.19x better than LLC. The results also show that the SI of a dataset almost has no impact on the LLC's CF. For MTD with SI=10m, MDB at ϵ=0% provides slightly lower compression than LLC due to the very small size of the dataset where there is an overhead from MDB's segment metadata. The results for WTM (Appendix B) are similar to those for MTD.

We answer **RQ1.3** as follows. *MDB provides the best CF for datasets with short SI* such as PCD. For datasets with higher SI where the variability between the values is high, MDB's compression becomes less effective and the impact of ϵ decreases when the SI is high.

5.2 Transfer Efficiency

We now evaluate how many values can be transferred from edge to cloud by the solutions when the bandwidth is limited to 512 Kbit/s (Challenge 2). The challenge is exacerbated for high frequent data and thus we use PCD. We do the experiment with 2 days of data. For AGG, we use SI=1.05s (i.e., 7x aggregation) that has the lowest compression error. For LLC and AGG, we implemented a Java program for ingesting data points into Apache ORC format. The transfer to the cloud is done by scp. For IoTDB, we created a Java program that ingests the data using the recommended native API on the edge.

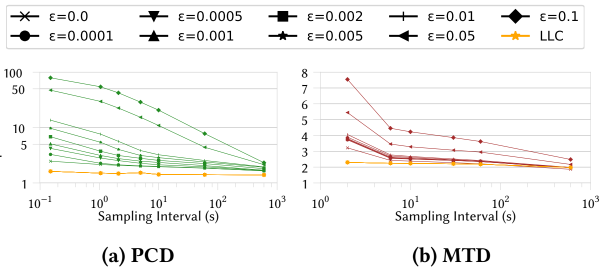


Figure 8: Impact of SI on MDB's CF for PCD and MTD.

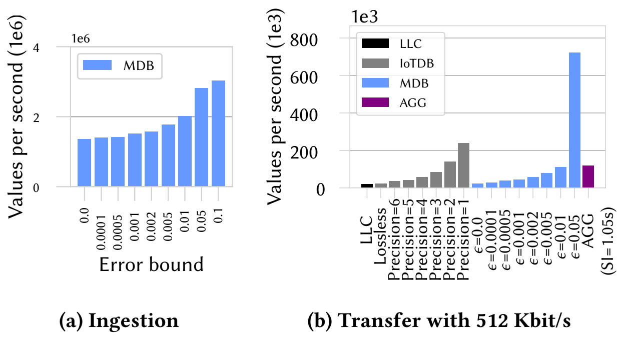


Figure 9: Values ingested and transferred per second for PCD.

The data is transferred to the cloud using IoTDB's Pipe feature with an `iotdb-thrift-async-connector` which is recommended for high transfer performance [27]. For all four solutions, we thus both ingest and transfer data. For LLC and AGG, the time for ingestion is negligible. MDB needs more time, but as shown in Figure 9a, MDB ingests from 1.4 to 3 million values/second with higher speeds for higher error bounds where the segments get longer. This is much faster than what can be transferred over the network.

Figure 9b shows the number of values handled per second when the ingested data is also transferred from edge to cloud. With LLC around 19,000 values are transferred per second. For ϵ=0%, MDB transfers 1.2x faster than LLC. IoTDB with lossless compression transfers 1.05x more values than MDB with ϵ=0%. This is because IoTDB can ingest and transfer in parallel while MDB cannot do this yet. When ϵ>0%, MDB transfers more values per second and this is correlated with the increase in CF shown in Section 5.1. AGG transfers 1.05x more values per second than MDB with ϵ=1%. AGG, however, produces unbounded errors that are many orders of magnitude higher than MDB and thus fails in terms of Challenge 4. With ϵ=5% MDB transfers 38x, 6x and 3x more values than LLC, AGG and IoTDB with precision=1, respectively, and with ϵ=10% MDB transfers 1.2e+6 values per second, i.e., 10x and 5x more than AGG and IoTDB with precision=1, respectively (not shown in Figure 9(b)). With an increase in bandwidth from 512 Kbit/s to 2.5 Mbit/s we saw similar, but ~ 5x higher throughput.

We answer **RQ2** as follows. LLC, can transfer around 19,000 values per second with 512 Kbit/s bandwidth while *MDB at ϵ=0% can transfer 1.2x more values than LLC.* In addition, *MDB can transfer even more values through the use of error-bounded lossy compression.* As an example, MDB's with ϵ=1% can transfer *6x more values than LLC* with 512 Kbit/s bandwidth. MDB with ϵ=1% *matches the transfer efficiency of AGG and has up to ~12 orders of magnitude better data quality.* Compared to IoTDB's lossless compression, MDB with ϵ=0% has a slightly lower transfer rate, but a higher CF and is thus *as good as a state-of-the-art solution for lossless compression.* For lossy compression, MDB with ϵ=5% *transfers 5x more and compresses 1.8x better* than IoTDB with precision=1. The overhead of ingestion is insignificant compared to the transfer time.

5.3 Data Quality

In this section, we measure the compression error of MDB using MAPE. We also measure the amount of values represented with no error by MDB. Finally, we analyze the impact of lossy compression on the quality of downstream analytics for the different solutions.

5.3.1 Compression Error. Figures 10a-10b show the distribution of MAPE for all signals of PCD and MTD, while the results for WTM are provided in D. Whiskers of box plots represent the minimum and maximum values. Whiskers that span all the way down to the x axis represent zero. PCD in comparison to the other datasets shows a higher MAPE. This is related to the higher use of PMC and Swing to compress PCD for all ϵ. In addition, lower MAPE for MTD and WTM is due to the significantly higher use of Gorilla even when we allow for some error. At ϵ=0.01% the maximum MAPE for PCD is 0.47x of ϵ, while for MTD and WTM the maximum MAPEs are 0.15x and 0.11x of ϵ, respectively. PCD's lowest MAPE is 0 for all ϵ meaning that a particular signal (*PowerLowerLimit*) is represented losslessly for all ϵ. Also certain periods when park operation is halted and no power is produced lead to the generation of constant values for most signals. This shows the effectiveness of MDB with constant signals. As we increase ϵ, there is a decrease of PCD's MAPE in relation to ϵ. At ϵ=0.1%, PCD's highest MAPE is 0.42x of ϵ, while at ϵ=1% and 10%, it decreases to 0.37x and 0.29x, respectively. Contrary to PCD, the difference between ϵ and compression error gradually decreases for MTD (Figure 10b) and WTM. As an example, the highest MAPE for MTD increases from 0.27x of ϵ at ϵ=0.1% to 0.34x of ϵ at ϵ=10%. The results for WTM are very similar. The increase in the compression error for MTD and WTM is due to the longer median segment length for all model types (Appendix C), while the decrease in the compression error for PCD happens because it is mostly represented by short PMC segments, which gives a good chance for approximating values with smaller error.

We also implemented a tool to determine the amount of values losslessly compressed by MDB [2]. Figure 10c shows that with ϵ=0.01%, MDB represents 71.9%–95.0% of the values losslessly, while 17.1%–34.3% of the values are compressed losslessly with ϵ=10%.

We answer **RQ3.1** as follows. With our datasets *MDB preserves the data quality even better than the guaranteed pointwise* ϵ. The model types have different impacts on the data quality. Higher use of Swing results in higher MAPE than PMC, while Gorilla compresses losslessly. Thus, PCD has a lower MAPE at higher error bounds than the other datasets. However, among all datasets and error bounds, the MAPE is less than half of the ϵ. Thus, *MDB preserves data quality much better than the* ϵ*, which allows for compressing with higher error bounds to maximize the compression effectiveness.*

5.3.2 Downstream Analytics. MDB provides efficient integration with the data science libraries like NumPy and pandas. Data scientists can query both the edge nodes and cloud nodes for scalable analytics using the Apache Arrow Flight interface. The data is returned in Apache Arrow format which can easily be converted to NumPy arrays and pandas dataframes. Through Apache Spark SQL, MDB provides full expressive power for advanced OLAP queries such as rollup, cube by, window functions and grouping sets. MDB efficiently computes simple aggregates and aggregates in the time

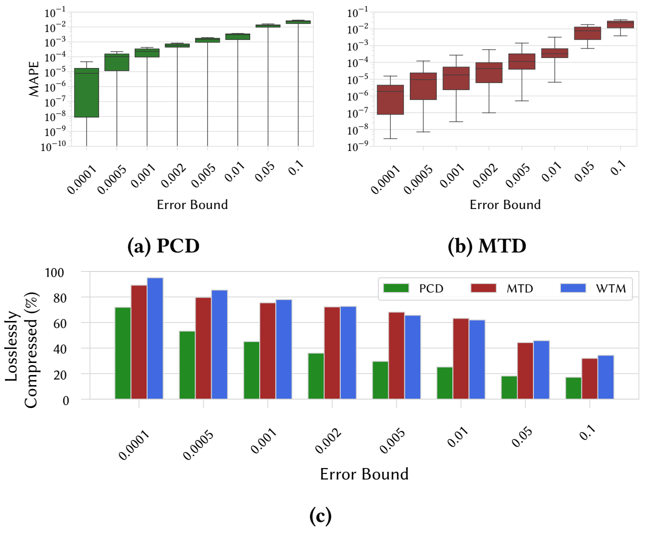


Figure 10: (a-b) MAPE of PCD and MTD for different ϵ. (c) Amount of values compressed with zero error for all datasets.

dimension directly on compressed segments [34]. In [52], MDB is used as a part of the MORE (Management of Real-time Energy data) platform [1]. The study uses the MTD for evaluation and MDB is used for providing lossily compressed data for Incremental Machine Learning (IML) models to detect so-called yaw misalignment in real-time. Similarly, in [12], MDB's implementation of PMC and Swing along with the dataset WTM are used for time series forecasting.

5.3.3 Downstream Analytics Accuracy. We evaluate the impact of lossy compression on MIN, AVG and STDEV aggregations for each signal in each dataset. Results for COUNT, MAX and SUM aggregations are available in the extended version (Appendix D). We do the aggregations for each signal and compute the Relative Query Error (RQE) as $|(Q - Q')/Q|$ where Q is the query result from the original dataset and Q' is the query result from the decompressed dataset. If $Q = Q'$, then RQE is 0. We ignore cases when $Q = 0 \neq Q'$ to avoid division by zero. For MIN we use the predicate `signal > 0` (and refer to it as MIN*) as MDB and IoTDB represent 0 losslessly. Results for MIN without a predicate are given in Appendix D. Table 5 shows the median and max RQEs for MIN*, AVG and STDEV over all signals for each dataset. Due to space constraints, we show only results for MDB, IoTDB and AGG that are comparable by CF. The remaining results are available in D. COUNT is exact in MDB and thus its RQEs for AVG and SUM are identical, while for AGG, COUNT and SUM can be computed using the SI. IoTDB's results for SUM are also very low. All solutions have low RQEs for MAX (Appendix D) as all results are higher than 0 and outliers tend to appear consecutively.

With MDB, all queries generate lower RQE than the ϵ and the median RQE is much smaller than the ϵ. Similarly, IoTDB provides low RQEs for AVG and STDEV, while for MIN*, its encoding format (i.e., TS_2DIFF) represents small values as 0 due to rounding to the configured precision. This results in very large RQEs. As an example, for PCD with precision=1, IoTDB's lowest possible value for MIN* is 0.1, while querying the original dataset returns small

Query	*SELECT MIN (signal) WHERE signal > 0*				*SELECT AVG (signal)*				*SELECT STDEV (signal)*			
	Median(error)		*Max(error)*		*Median(error)*		*Max(error)*		*Median(error)*		*Max(error)*	
MDB ϵ	*0.01 (CF=14x)*	*0.05 (CF=47x)*	*0.01 (CF=14x)*	*0.05 (CF=47x)*	*0.01 (CF=14x)*	*0.05 (CF=47x)*	*0.01 (CF=14x)*	*0.05 (CF=47x)*	*0.01 (CF=14x)*	*0.05 (CF=47x)*	*0.01 (CF=14x)*	*0.05 (CF=47x)*
RQE	4e-9	4e-9	2e-8	2e-8	3e-4	0.005	0.0008	0.007	8e-5	3e-4	4e-4	0.001
IoTDB Precision	*2 (CF=13x)*	*1 (CF=26x)*	*2 (CF=13x)*	*1 (CF=26x)*	*2 (CF=13x)*	*1 (CF=26x)*	*2 (CF=13x)*	*1 (CF=26x)*	*2 (CF=13x)*	*1 (CF=26x)*	*2 (CF=13x)*	*1 (CF=26x)*
RQE	2.8e+12	2.8e+13	1.7e+36	1.7e+37	0.005	0.008	0.017	0.02	0.002	0.005	0.011	0.12
AGG SI	*1.05s (CF=10.5x)*	*4.95s (CF=50x)*	*1.05s (CF=10.5x)*	*4.95s (CF=50x)*	*1.05s (CF=10.5x)*	*4.95s (CF=50x)*	*1.05s (CF=10.5x)*	*4.95s (CF=50x)*	*1.05s (CF=10.5x)*	*4.95s (CF=50x)*	*1.05s (CF=10.5x)*	*4.95s (CF=50x)*
RQE	1.6	14	8.2e+10	1e+10	3e-6	5e-6	3e-4	6e-4	5e-6	1e-5	0.03	0.1

Query	*SELECT MIN (signal) WHERE signal > 0*				*SELECT AVG (signal)*				*SELECT STDEV (signal)*			
	Median(error)		*Max(error)*		*Median(error)*		*Max(error)*		*Median(error)*		*Max(error)*	
MDB ϵ	*0.05 (CF=5.5x)*	*0.1 (CF=7.5x)*	*0.05 (CF=5.5x)*	*0.1 (CF=7.5x)*	*0.05 (CF=5.5x)*	*0.1 (CF=7.5x)*	*0.05 (CF=5.5x)*	*0.1 (CF=7.5x)*	*0.05 (CF=5.5x)*	*0.1 (CF=7.5x)*	*0.05 (CF=5.5x)*	*0.1 (CF=7.5x)*
RQE	3e-8	3e-8	0.002	0.02	6e-4	0.003	0.004	0.01	7e-4	0.004	0.007	0.03
IoTDB Precision	*3 (CF=6x)*	*2 (CF=8x)*	*3 (CF=6x)*	*2 (CF=8x)*	*3 (CF=6x)*	*2 (CF=8x)*	*3 (CF=6x)*	*2 (CF=8x)*	*3 (CF=6x)*	*2 (CF=8x)*	*3 (CF=6x)*	*2 (CF=8x)*
RQE	45	268	1.9e+6	1.9e+7	2e-6	5e-6	0.002	0.003	5e-7	5.6e-6	0.002	0.004
AGG SI	*6s (CF=8x)*	*10s (CF=13x)*	*6s (CF=8x)*	*10s (CF=13x)*	*6s (CF=8x)*	*10s (CF=13x)*	*6s (CF=8x)*	*10s (CF=13x)*	*6s (CF=8x)*	*10s (CF=13x)*	*6s (CF=8x)*	*10s (CF=13x)*
RQE	0.6	3	656382	673499	9e-5	2e-4	0.008	0.02	8e-4	0.002	0.04	0.05

Table 4: Relative query errors (in %) of OLAP queries on PCD (left) and MTD (right).

values like 3.5e-15. This results in very large RQEs such as 2.8e+13. MDB's pointwise ϵ provides finer control over the error for lossy compression. AGG suffers from a similar issue as IoTDB with MIN*. For example, with SI=4.95s for PCD it has 1e+10 RQE, while for AVG and STD, it produces accurate results similar to IoTDB and MDB. Table 5 also shows the CFs of each method and we can see that for PCD, the dataset with the highest frequency, MDB hits the sweet spot between compression and query accuracy by providing the best overall quality for OLAP queries and better compression than IoTDB's lossy compression. For MTD, MDB provides the best overall query accuracy and comparable compression to IoTDB. AGG can be used for achieving a very high CF, however, it also results in unbounded errors for both data quality and OLAP query accuracy. IoTDB provides better CF for a dataset with higher SI and its lossy compression's query accuracy highly depends on the configured precision limit and the range of values in the dataset.

The impact of error-bounded lossy compression methods including PMC and Swing on the accuracy of state-of-the-art time series forecasting models is studied in [12]. The study uses MDB's implementation of PMC and Swing. The study finds that error-bounded lossy compression can be performed with an ϵ up to 30% for PMC and 25% for Swing before significantly reducing the accuracy of forecasting models. The results in [52] show that the IML models trained with MDB's lossily compressed data using $2\% \leq \epsilon \leq 10\%$ provide 1.05x and 1.26x better f1-scores than the models trained with 400x (i.e., 1min) and 4000x (i.e., 10min) AGG data, respectively. 10min AGG data is an industry standard method of data preprocessing for yaw misalignment detection [20]. Compressing with ϵ=2%, the lowest ϵ used in the study, provides the same f1-score as 2x AGG. As for compression, ϵ=10% provides similar compression to 400x AGG. Both studies also mention that in some cases, the use of lossy compression even improves the accuracy of the models.

We answer **RQ3.2** as follows. MDB's error-bounded lossy compression *produces much lower RQE* than the ϵ for all our datasets and queries. All solutions perform well for aggregate queries with AVG and STDEV. For MIN*, MDB's error-bounded lossy compression produces significantly better results than IoTDB and AGG. MDB can efficiently be integrated into scalable data science infrastructures as manifested in [52]. Studies [12, 52] show that error-bounded lossily compressed data can *effectively be used for time series forecasting and yaw missalignment detection in real-life scenarios.*

6 RELATED WORK

Many time series compression methods have been proposed, this survey [10] splits them into five categories: *Dictionary-based*, *Functional Approximation*, *Autoencoders*, *Sequential methods*, and *Others*.

Dictionary-based methods like [38, 40, 43, 47] build dictionaries of subsequences and represent time series using them. Achieving high dictionary search speed and low size are the main challenges.

Functional Approximation methods like [13, 22, 23, 36, 57] split time series into segments and approximate each segment using a function of time. They can be used for online compression as they do not have a training phase and are also computationally efficient.

Autoencoders like [21, 42, 48, 58, 59] are related neural networks with an Encoder and a Decoder. They cannot do online compression on the edge as they are resource intensive [10] and require a GPU.

Sequential methods like [5, 8, 41, 46, 49, 50] apply several simple compression methods sequentially. They can be used online as they do not have a training phase and are also computationally efficient.

Others like [15, 25] are methods that do not fit into a category.

An evaluation [24] of model-based compression methods [4, 6, 7, 14, 17, 37, 39] for sensor data, found that APCA [37] and SF [14] provided the best compression factor for time series with little noisy, while APCA [37] and GAMPS [17] were good for noisy time series. They also found that a dynamic segment size is very important to achieve a high compression factor and very low compression error.

An evaluation [56] of lossless time series compression methods [5, 9, 19, 46, 51, 55] in IoTDB [53, 54], found that TS_2DIFF [55] achieves higher compression factors for datasets with a large delta mean, while Gorilla [46] achieves higher compression factors for datasets with a small delta mean and value variance. Gorilla also provides lower compression and decompression time than TS_2DIFF.

While these studies evaluate lossy and lossless compression methods, they do not combine them or use big real-life datasets.

7 CONCLUSION

We show that ModelarDB (MDB) addresses all four challenges of managing high frequency wind turbine data across edge and cloud: *Limited Hardware*, *Limited Bandwidth*, *High Storage Costs*, and *Low Data Quality After Compression.* Compared to solutions used in industry and with real-life high frequent wind turbine datasets, MDB has a higher compression factor and transfer efficiency than LLC and matches AGG's compression factor and transfer efficiency but adds orders of magnitude less error. Compared to IoTDB, MDB has a higher compression factor but slightly worse transfer efficiency with losless compression. However, with lossy compression, MDB has a higher compression factor and transfer efficiency than IoTDB for the very high frequency dataset.

We found that MDB's focus on compression factor instead of compression speed is beneficial due to the limited bandwidth from the edge to the cloud, and that MDB provides a good compromise between compression factor and data quality since the models generally have much less error than the error bound. Thus, MDB is excellent at managing high frequency wind turbine data across the edge and cloud.

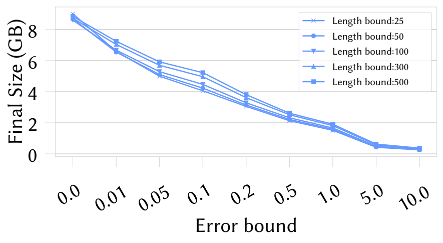
In future work, we will design new model types and improve the model fitting strategy by exploiting properties of the time series. Both will be added to a full reimplementation of ModelarDB [45].

ACKNOWLEDGMENTS

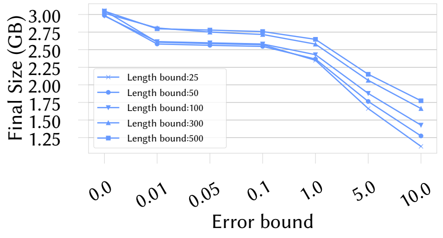
Funded by EU Horizon 2020 grant number 955895 and 957345. We thank Siemens Gamesa Renewable Energy (SGRE) and our anonymous industry partner for access to real-life data. We thank Senior Data Scientist Emil Hedevang at SGRE for insightful comments.

A LENGTH BOUND AND BATCH SIZE OPTIMIZATION

In order to find the *length bound* that provides the highest CF, we ingest the datasets PCD and MTD using the following length bound parameters: *25, 50, 100, 300 and 500.* The results for PCD and MTD can be seen in Figures 11a-11b. We can see that for PCD, the length bound of 25 and 50 provide the best CF for all error bounds. For MTD, the length bound of 50 values provides the best CF for ϵ 0%, 0.01%, 0.05%, 0.1% and 1%, while the length bound of 25 values provides the best CF for ϵ 5% and 10%. Thus, we use the length bound 50 that provides the best CF for most datasets and error bounds. To find the best batch size that provides the shortest ingestion time, we ingest datasets using the following batch sizes: *500, 1000, 5000, 10000, 50000 and 100000* and error bounds: 0%, 1%, 5% and 10%. As batch size only impacts the ingestion time, we perform the experiment only for PCD that is the only dataset used for transfer efficiency experiment in Section 5.2. The experiment is performed in the same experimental setup as in experiments in Section 5.2. Figure 12a shows the ingestion time results. We can see that MDB with batch size of 1000 ingests most data points when $0\% \leq \epsilon \leq 1\%$ and with batch size 50,000 it ingests most data points when $5\% \leq \epsilon \leq 10\%$. Given that the most of the error bounds used in our experiments are in the range of $0\% \leq \epsilon \leq 1\%$, we use the batch size 1000 in our experiments. To conclude, the length bound 50 provides the best compression effectiveness in 85% of combinations and batch size of 1000 provides the best ingestion time in 50% of combinations.

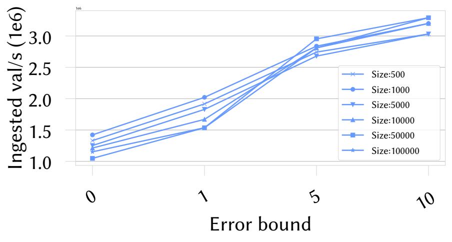


(a) PCD



(b) MTD

Figure 11: CFs for different length bound parameters for PCD and MTD



(a) PCD

Figure 12: Values ingested per second (in 1e6) for different batch size parameters for PCD and on the edge.

B SECTION 5.1 AND 5.3 FIGURES FOR MTD AND WTM

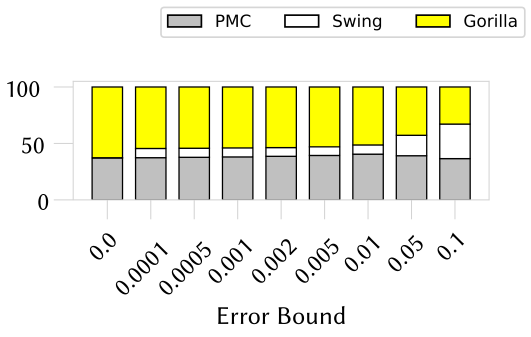


Figure 13: Distribution of values compressed by MDB's model types for WTM

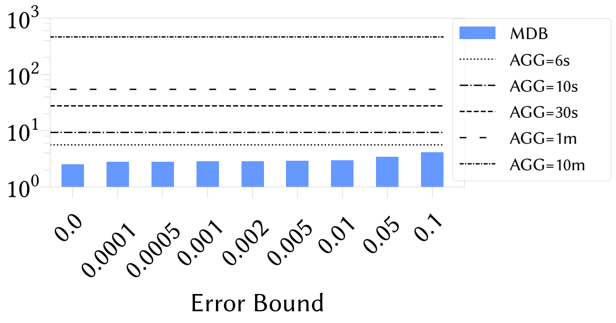


Figure 14: MDB (bars) and AGG (lines) CFs for WTM. x-axis (i.e., error bound) is only used for MDB. AGG is independent of the error bound.

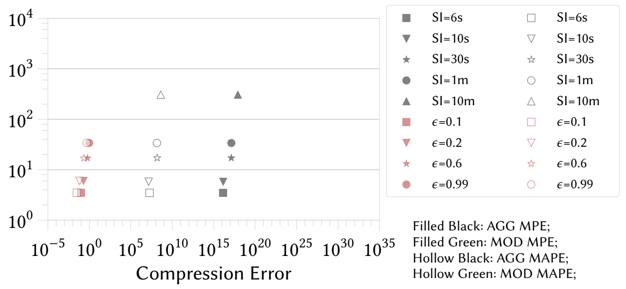


Figure 15: CFs and resulting errors for MDB and AGG on MTD.

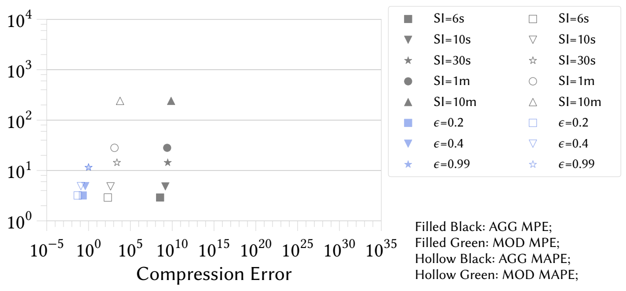


Figure 16: CFs and resulting errors for MDB and AGG on WTM.

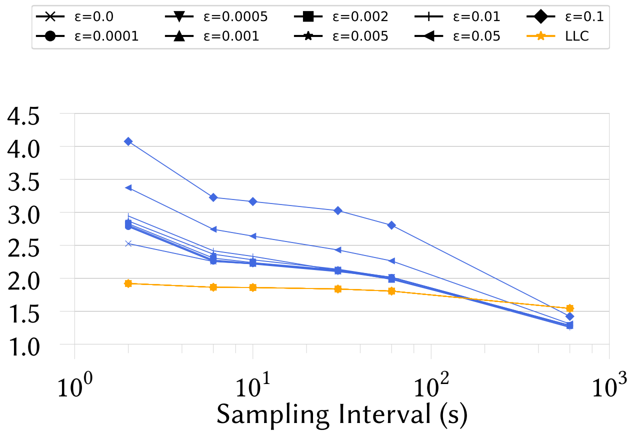


Figure 17: Impact of SI on MDB's CF for WTM.

C MEAN AND MEDIAN SIZE OF SEGMENTS

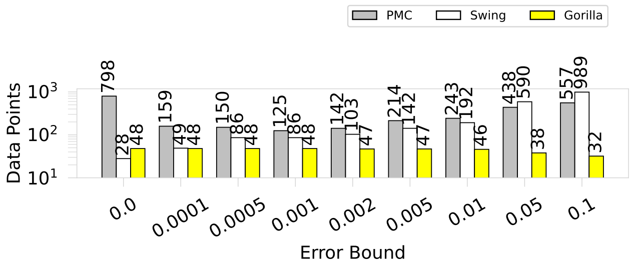


Figure 18: Mean segment length of each model type for PCD

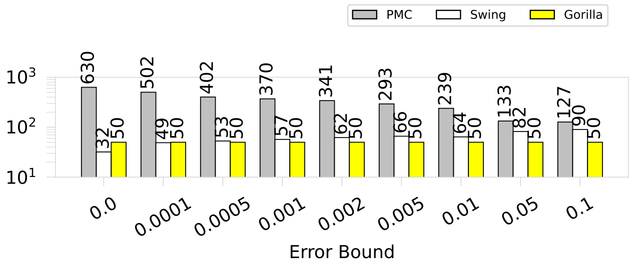


Figure 19: Mean segment length of each model type for MTD

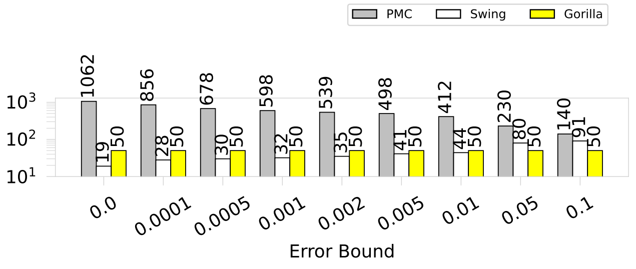


Figure 20: Mean segment length of each model type for WTM

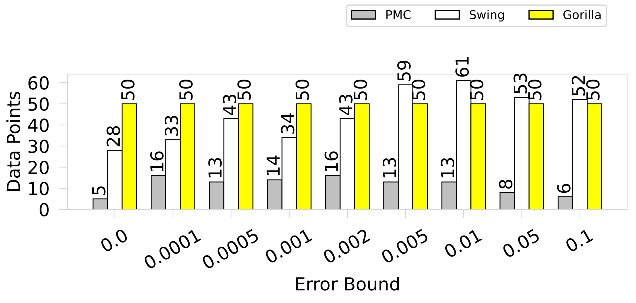


Figure 21: Median segment length of each model type for PCD

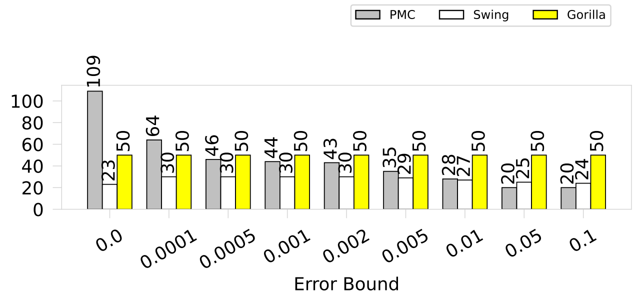


Figure 22: Median segment length of each model type for MTD

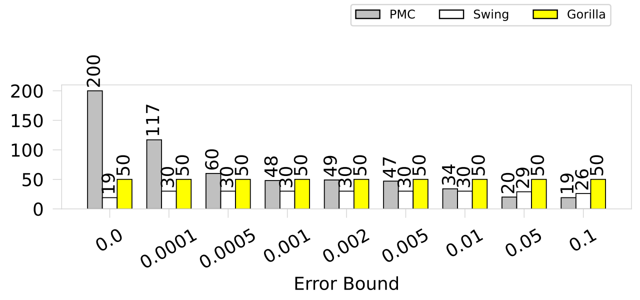


Figure 23: Median segment length of each model type for WTM

D DOWNSTREAM ANALYTICS ACCURACY

Query	SELECT MIN (signal) WHERE signal > 0									
	Median(error)					Max(error)				
MDB (ϵ) / Dataset	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79
PCD	2e-9	4e-9	4e-9	4e-9	5e-9	2e-8	2e-8	2e-8	2e-8	2e-8
MTD	2e-8	3e-8	2e-8	3e-8	3e-8	5e-8	5e-8	0.002	0.002	0.02
IoTDB (Precision) / Dataset	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26
PCD	2.8e+9	2.8e+10	2.8e+11	2.8e+12	2.8e+13	1.7e+33	1.7e+34	1.7e+35	1.7e+36	1.7e+37
MTD	0.04	0.9	45	268	1865	763362	754973	1.9e+6	1.9e+7	1.9e+8
AGG (SI) / Dataset	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572
PCD	1.6	14	23	5	112	8.2e+10	1e+10	1.2e+11	1.1e+13	1.8e+13
AGG (SI) / Dataset	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707
MTD	0.6	3	7.5	14	50	656382	673499	2.3e+6	3.5e+6	5.3e+7

Query	SELECT AVG (signal)									
	Median(error)					Max(error)				
MDB (ϵ) / Dataset	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79
PCD	6e-6	8e-6	3e-4	0.005	0.01	2e-5	5e-5	0.0008	0.007	0.002
MTD	9e-7	2e-6	5e-5	6e-4	0.003	3e-5	1e-4	8e-4	0.004	0.01
IoTDB (Precision) / Dataset	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26
PCD	4e-7	4e-6	3e-4	0.005	0.008	0.01	0.015	0.016	0.017	0.02
MTD	0	8e-8	2e-6	5e-6	6e-5	4e-6	3e-5	0.002	0.003	0.02
AGG (SI) / Dataset	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572
PCD	3e-6	5e-6	5e-6	3.6e-5	5e-4	3e-4	6e-4	9e-4	0.002	0.005
AGG (SI) / Dataset	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707
MTD	9e-5	2e-4	7e-4	0.001	0.005	0.008	0.02	0.06	0.07	0.2

Query	SELECT STDEV (signal)									
	Median(error)					Max(error)				
MDB (ϵ) / Dataset	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79
PCD	1e-5	2e-5	8e-5	3e-4	4e-4	2.3e-5	4e-5	4e-4	0.001	0.002
MTD	2e-6	4e-6	1e-4	7e-4	0.004	4e-5	6e-5	0.001	0.007	0.03
IoTDB (Precision) / Dataset	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26
PCD	1e-7	1e-6	2e-4	0.002	0.005	0.01	0.01	0.011	0.011	0.12
MTD	0	8e-8	5e-7	5.6e-6	5e-5	1e-6	1e-5	0.002	0.004	0.01
AGG (SI) / Dataset	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572
PCD	5e-6	1e-5	1e-5	1e-4	7e-4	0.03	0.1	0.3	1.1	4.1
AGG (SI) / Dataset	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707
MTD	8e-4	0.002	0.005	0.012	0.04	0.04	0.05	0.08	0.1	0.12

Query	SELECT MIN (signal)									
	Median(error)					Max(error)				
MDB (ϵ) / Dataset	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79
PCD	0	0	0	0	0	3e-4	5e-4	0.004	0.047	0.07
MTD	0	0	0	0	0	4.9e-4	9e-4	0.009	0.049	0.099
IoTDB (Precision) / Dataset	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26
PCD	0	0	0	0	0	0	0	0	0	2e-3
MTD	0	0	0	0	0	0	0	0	0	0.02
AGG (SI) / Dataset	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572
PCD	0.08	0.2	0.21	0.21	0.23	0.6	1	1	1	1
AGG (SI) / Dataset	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707
MTD	0	0	0	0	0	223	246	955	1396	13169

Query	SELECT MAX (signal)									
	Median(error)					Max(error)				
MDB (ϵ) / Dataset	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79
PCD	0	0	4e-4	0.019	0.067	4e-4	7e-4	0.009	0.049	0.099
MTD	0	0	0	0.014	0.06	4.9e-4	9e-4	0.009	0.049	0.099
IoTDB (Precision) / Dataset	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26
PCD	0	0	0	0	0	0	0	0	0	0
MTD	0	0	0	0	0	0	0	0	0	0
AGG (SI) / Dataset	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572
PCD	0.003	0.006	0.007	0.016	0.2	0.06	0.14	0.17	0.19	0.3
AGG (SI) / Dataset	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707
MTD	6e-4	2e-3	8e-4	0.02	0.05	0.14	0.3	0.5	0.5	0.6

Query	SELECT SUM (signal)									
	Median(error)					Max(error)				
MDB (ϵ) / Dataset	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79	0.0005 CF=4.2	0.001 CF=5	0.01 CF=14	0.05 CF=47	0.1 CF=79
PCD	6e-6	8e-6	3e-4	0.005	0.01	2e-5	5e-5	0.0008	0.007	0.002
MTD	9e-7	2e-6	5e-5	6e-4	0.003	3e-5	1e-4	8e-4	0.004	0.01
IoTDB (Precision) / Dataset	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26	5 CF=3.7	4 CF=5	3 CF=7.4	2 CF=13	1 CF=26
PCD	6e-7	1e-6	8e-7	3e-6	2e-6	6e-6	8e-6	4e-6	5e-5	2e-5
MTD	0	7e-8	3e-7	4e-7	3e-6	2e-6	2e-6	3e-6	4e-6	3e-4
AGG (SI) / Dataset	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572	1.05s CF=10.5	4.95s CF=50	10.05s CF=95	1m CF=563	10m CF=5572
PCD	3e-6	5e-6	5e-6	3.6e-5	5e-4	3e-4	6e-4	9e-4	0.002	0.005
AGG (SI) / Dataset	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707	6s CF=8	10s CF=13	30s CF=39	1m CF=77	10m CF=707
MTD	9e-5	2e-4	7e-4	0.001	0.005	0.008	0.02	0.06	0.07	0.2

Table 5: Relative query errors (in %) of OLAP queries on PCD and MTD. COUNT is exact for both MDB and IoTDB and for AGG it can be computed by the number of aggregated values.

This section illustrates the complete results of running the OLAP queries with lossy compressed data by MDB, IoTDB and AGG. We run the following queries: MIN, MIN(signal) WHERE signal > 0, MAX, AVG, SUM and STDEV for PCD and MTD.

REFERENCES

[1] European Union's Horizon 2020. 2020. MORE: Management of Real-time Energy Data. https://www.more2020.eu/. Viewed: 2024-07-12.

[2] Abduvoris Abduvakhobov. 2023. ModelarDB-Analyzer. https://github.com/aabduvakhobov/ModelarDB-Analyzer. Viewed: 2024-07-12.

[3] Inc. Amazon Web Services. 2024. Amazon S3 pricing. https://aws.amazon.com/s3/pricing/. Viewed: 2024-07-12.

[4] Alexandru Arion, Hoyoung Jeung, and Karl Aberer. 2011. Efficiently Maintaining Distributed Model-Based Views on Real-Time Data Streams. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*. IEEE, Houston, TX, USA, 1–6. https://doi.org/10.1109/GLOCOM.2011.6133764

[5] Davis Blalock, Samuel Madden, and John Guttag. 2018. Sprintz: Time Series Compression for the Internet of Things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3, Article 93 (sep 2018), 23 pages. https://doi.org/10.1145/3264903

[6] Chiranjeeb Buragohain, Nisheeth Shrivastava, and Subhash Suri. 2007. Space Efficient Streaming Algorithms for the Maximum Error Histogram. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, Istanbul, Turkey, 1026–1035. https://doi.org/10.1109/ICDE.2007.368961

[7] Yuhan Cai and Raymond Ng. 2004. Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (Paris, France) *(SIGMOD '04)*. Association for Computing Machinery, New York, NY, USA, 599–610. https://doi.org/10.1145/1007568.1007636

[8] Giuseppe Campobello, Antonino Segreto, Sarah Zanafi, and Salvatore Serrano. 2017. RAKE: A Simple and Efficient Lossless Compression Algorithm for the Internet of Things. In *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, Kos, Greece, 2581–2585. https://doi.org/10.23919/EUSIPCO.2017.8081677

[9] Giuseppe Campobello, Antonino Segreto, Sarah Zanafi, and Salvatore Serrano. 2017. RAKE: A Simple and Efficient Lossless Compression Algorithm for the Internet of Things. In *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, Kos, Greece, 2581–2585. https://doi.org/10.23919/EUSIPCO.2017.8081677

[10] Giacomo Chiarot and Claudio Silvestri. 2023. Time Series Compression Survey. *ACM Comput. Surv.* 55, 10, Article 198 (feb 2023), 32 pages. https://doi.org/10.1145/3560814

[11] Carlos Enrique Muniz Cuza. 2023. WTM (Wind) dataset. https://github.com/cmcuza/EvalImpLSTS/tree/main/data/raw/Wind. Viewed: 2024-07-12.

[12] Carlos Enrique Muniz Cuza, Jensen Søren Kejser, Brusokas Jonas, Ho Nguyen, and Pedersen Torben Bach. 2024. Evaluating the Impact of Error-Bounded Lossy Compression on Time Series Forecasting. In *Proceedings of the 27th International Conference on Extending Database Technology, 2024*. OpenProceedings.org, Paestum, Italy, 651–663. https://doi.org/10.48786/edbt.2024.56

[13] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. 2015. A Time-Series Compression Technique and Its Application to the Smart Grid. *The VLDB Journal* 24 (2015), 193–218. https://doi.org/10.1007/s00778-014-0368-8

[14] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. 2009. Online Piece-Wise Linear Approximation of Numerical Streams with Precision Guarantees. *Proceedings of the VLDB Endowment* 2, 1 (aug 2009), 145–156. https://doi.org/10.14778/1687627.1687645

[15] Eugene Fink and Harith Suman Gandhi. 2011. Compression of Time series by Extracting Major Extrema. *Journal of Experimental & Theoretical Artificial Intelligence* 23, 2 (2011), 255–270. https://doi.org/10.1080/0952813X.2010.505800

[16] Apache Software Foundation. 2023. Apache Arrow. https://arrow.apache.org/docs/11.0/. Viewed: 2024-07-12.

[17] Sorabh Gandhi, Suman Nath, Subhash Suri, and Jie Liu. 2009. GAMPS: Compressing Multi Sensor Data by Grouping and Amplitude Scaling. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Providence, Rhode Island, USA) *(SIGMOD '09)*. Association for Computing Machinery, New York, NY, USA, 771–784. https://doi.org/10.1145/1559845.1559926

[18] Arnav Gohil, Anshul Shroff, Arnav Garg, and Shailender Kumar. 2022. A Compendious Research on Big Data File Formats. In *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, Madurai, India, 905–913. https://doi.org/10.1109/ICICCS53718.2022.9788141

[19] Solomon W. Golomb. 1966. Run-Length Encodings (corresp.). *IEEE Transactions on Information theory* 12, 3 (1966), 399–401.

[20] E Gonzalez, B Stephen, D Infield, and J J Melero. 2017. On the use of high-frequency SCADA data for improved wind turbine performance monitoring. *Journal of Physics: Conference Series* 926, 1 (November 2017), 012009. https://doi.org/10.1088/1742-6596/926/1/012009

[21] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2021. DZip: improved general-purpose loss less compression based on novel neural network modeling. In *2021 Data Compression Conference (DCC)*. IEEE, Snowbird, UT, USA, 153–162. https://doi.org/10.1109/DCC50243.2021.00023

[22] S. Edward III Hawkins and Edward Hugo Darlington. 2012. *Algorithm for Compressing Time-Series Data.* Technical Report. NASA Tech Briefs.

[23] Yuh-Ming Huang and Ja-Ling Wu. 1999. A Refined Fast 2-D Discrete Cosine Transform Algorithm. *IEEE Transactions on Signal Processing* 47, 3 (1999), 904–907. https://doi.org/10.1109/78.747801

[24] Nguyen Quoc Viet Hung, Hoyoung Jeung, and Karl Aberer. 2013. An Evaluation of Model-Based Approaches to Sensor Data Compression. *IEEE Transactions on Knowledge and Data Engineering* 25, 11 (2013), 2434–2447. https://doi.org/10.1109/TKDE.2012.237

[25] T. Inamura, H. Tanie, and Y. Nakamura. 2003. Keyframe Compression and Decompression for Time Series Data Based on the Continuous Hidden Markov Model. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Vol. 2. IEEE, Las Vegas, NV, USA, 1487–1492 vol.2. https://doi.org/10.1109/IROS.2003.1248854

[26] Apache IoTDB. 2024. Encoding and Compression. https://iotdb.apache.org/UserGuide/V1.2.x/Basic-Concept/Encoding-and-Compression.html. Viewed: 2024-07-12.

[27] Apache IoTDB. 2024. IoTDB Data Sync. https://iotdb.apache.org/UserGuide/V1.2.x/User-Manual/Data-Sync.html. Viewed: 2024-07-12.

[28] Apache IoTDB. 2024. Quick Start. https://iotdb.apache.org/UserGuide/latest/QuickStart/QuickStart.html. Viewed: 2024-07-12.

[29] Søren Kejser Jensen and Christian Thomsen. 2023. Holistic Analytics of Sensor Data from Renewable Energy Sources: A Vision Paper. In *27th European Conference on Advances in Databases and Information Systems*. Springer Cham, Barcelona, Spain, 360–366. https://doi.org/10.1007/978-3-031-42941-5_31

[30] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. [n.d.]. Time Series Management Systems: A 2022 Survey. In *Data Series Management and Analytics (Forthcoming)*, Themis Palpanas and Kostas Zoumpatianos (Eds.). ACM. Preprint available at: https://vbn.aau.dk/da/publications/time-series-management-systems-a-2022-survey.

[31] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2017. Time Series Management Systems: A Survey. *IEEE Trans. Knowl. Data Eng.* 29, 11 (2017), 2581–2600. https://doi.org/10.1109/TKDE.2017.2740932

[32] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2018. ModelarDB: Modular Model-Based Time Series Management with Spark and Cassandra. *Proceedings of the VLDB Endowment* 11, 11 (jul 2018), 1688–1701. https://doi.org/10.14778/3236187.3236215

[33] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2019. Demonstration of ModelarDB: Model-Based Management of Dimensional Time Series. In *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*. ACM, Amsterdam, Netherlands, 1933–1936. https://doi.org/10.1145/3299869.3320216

[34] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2021. Scalable Model-Based Management of Correlated Dimensional Time Series in ModelarDB+. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, Chania, Greece, 1380–1391. https://doi.org/10.1109/ICDE51399.2021.00123

[35] Søren Kejser Jensen, Christian Thomsen, and Torben Bach Pedersen. 2023. *ModelarDB: Integrated Model-Based Management of Time Series from Edge to Cloud.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–33. https://doi.org/10.1007/978-3-662-66863-4_1

[36] Samsul Ariffin Abdul Karim, Mohd Hafizi Kamarudin, Bakri Abdul Karim, Mohammad Khatim Hasan, and Jumat Sulaiman. 2011. Wavelet Transform and Fast Fourier Transform for Signal Compression: A Comparative Study. In *2011 International Conference on Electronic Devices, Systems and Applications (ICEDSA)*. IEEE, Kuala Lumpur, Malaysia, 280–285. https://doi.org/10.1109/ICEDSA.2011.5959031

[37] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *SIGMOD Rec.* 30, 2 (may 2001), 151–162. https://doi.org/10.1145/376284.375680

[38] Abdelouahab Khelifati, Mourad Khayati, and Philippe Cudré-Mauroux. 2019. CORAD: Correlation-Aware Compression of Massive Time Series using Sparse Dictionary Coding. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, Los Angeles, CA, USA, 2289–2298. https://doi.org/10.1109/BigData47090.2019.9005580

[39] I. Lazaridis and S. Mehrotra. 2003. Capturing Sensor-Generated Time Series with Quality Guarantees. In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*. IEEE, Bangalore, India, 429–440. https://doi.org/10.1109/ICDE.2003.1260811

[40] Tuong Ly Le and Minh-Huan Vo. 2018. Lossless Data Compression Algorithm to Save Energy in Wireless Sensor Network. In *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*. IEEE, Ho Chi Minh City, Vietnam, 597–600. https://doi.org/10.1109/GTSD.2018.8595614

[41] Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2022. Chimp: Efficient Lossless Floating Point Compression for Time Series Databases. *Proceedings of the VLDB Endowment* 15, 11 (jul 2022), 3058–3070. https://doi.org/10.14778/3551793.3551852

[42] Yu Mao, Jingzong Li, Yufei Cui, and Jason Chun Xue. 2023. Faster and Stronger Lossless Compression with Optimized Autoregressive Framework. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 1–6. https://doi.org/10.1109/DAC56929.2023.10247866

[43] Alice Marascu, Pascal Pompey, Eric Bouillet, Michael Wurst, Olivier Verscheure, Martin Grund, and Philippe Cudre-Mauroux. 2014. TRISTAN: Real-Time Analytics on Massive Time Series Using Sparse Dictionary Compression. In *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, Washington, DC, USA, 291–300. https://doi.org/10.1109/BigData.2014.7004244

[44] ModelarData. 2021. ModelarDB-JVM. https://github.com/modelardata/modelardb. SHA: 81990207e8f1851ba0a6f2bc5a4609288c61358c, Viewed: 2024-07-12.

[45] ModelarData. 2023. ModelarDB-RS. https://github.com/ModelarData/ModelarDB-RS. Viewed: 2024-07-12.

[46] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. 2015. Gorilla: A Fast, Scalable, in-Memory Time Series Database. *Proceedings of the VLDB Endowment* 8, 12 (aug 2015), 1816–1827. https://doi.org/10.14778/2824032.2824078

[47] James Pope, Antonis Vafeas, Atis Elsts, George Oikonomou, Robert Piechocki, and Ian Craddock. 2018. An Accelerometer Lossless Compression Algorithm and Energy Analysis for IoT Devices. In *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, Barcelona, Spain, 396–401. https://doi.org/10.1109/WCNCW.2018.8368985

[48] Liang Qin and Jie Sun. 2023. Model Compression for Data Compression: Neural Network Based Lossless Compressor Made Practical. In *2023 Data Compression Conference (DCC)*. IEEE, Snowbird, UT, USA, 52–61. https://doi.org/10.1109/DCC55655.2023.00013

[49] Subhra J. Sarkar, Palash K. Kundu, and Gautam Sarkar. 2018. Development of Lossless Compression Algorithms for Power System Operational Data. *IET Generation, Transmission & Distribution* 12, 17 (2018), 4045–4052. https://doi.org/10.1049/iet-gtd.2018.5600

[50] Julien Spiegel, Patrice Wira, and Gilles Hermann. 2018. A Comparative Experimental Study of Lossless Compression Algorithms for Enhancing Energy Efficiency in Smart Meters. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, Porto, Portugal, 447–452. https://doi.org/10.1109/INDIN.2018.8471921

[51] Julien Spiegel, Patrice Wira, and Gilles Hermann. 2018. A Comparative Experimental Study of Lossless Compression Algorithms for Enhancing Energy Efficiency in Smart Meters. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, Porto, Portugal, 447–452. https://doi.org/10.1109/INDIN.2018.8471921

[52] Seshu Tirupathi, Dhaval Salwala, Giulio Zizzo, Ambrish Rawat, Mark Purcell, Søren Kejser Jensen, Christian Thomsen, Nguyen Ho, Carlos E. Muniz Cuza, Jonas Brusokas, Torben Bach Pedersen, Giorgos Alexiou, Giorgos Giannopoulos, Panagiotis Gidarakos, Alexandros Kalimeris, Stavros Maroulis, George Papastefanatos, Ioannis Psarros, Vassilis Stamatopoulos, and Manolis Terrovitis. 2022. Machine Learning Platform for Extreme Scale Computing on Compressed IoT Data. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, Osaka, Japan, 3179–3185. https://doi.org/10.1109/BigData55660.2022.10020540

[53] Chen Wang, Xiangdong Huang, Jialin Qiao, Tian Jiang, Lei Rui, Jinrui Zhang, Rong Kang, Julian Feinauer, Kevin A. McGrail, Peng Wang, Diaohan Luo, Jun Yuan, Jianmin Wang, and Jiaguang Sun. 2020. Apache IoTDB: Time-Series Database for Internet of Things. *Proceedings of the VLDB Endowment* 13, 12 (aug 2020), 2901–2904. https://doi.org/10.14778/3415478.3415504

[54] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache IoTDB: A Time Series Database for IoT Applications. *Proc. ACM Manag. Data* 1, 2, Article 195 (jun 2023), 27 pages. https://doi.org/10.1145/3589775

[55] Haoyu Wang and Shaoxu Song. 2022. Frequency Domain Data Encoding in Apache IoTDB. *Proceedings of the VLDB Endowment* 16, 2 (oct 2022), 282–290. https://doi.org/10.14778/3565816.3565829

[56] Jinzhao Xiao, Yuxiang Huang, Changyu Hu, Shaoxu Song, Xiangdong Huang, and Jianmin Wang. 2022. Time Series Data Encoding for Efficient Storage: A Comparative Analysis in Apache IoTDB. *Proceedings of the VLDB Endowment* 15, 10 (jun 2022), 2148–2160. https://doi.org/10.14778/3547305.3547319

[57] Jialing Zhang, Jiaxi Chen, Aekyeung Moon, Xiaoyan Zhuo, and Seung Woo Son. 2020. Bit-Error Aware Quantization for DCT-based Lossy Compression. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Waltham, MA, USA, 1–7. https://doi.org/10.1109/HPEC43674.2020.9286177

[58] Xia Zhao, Xiao Han, Weijun Su, and Zhen Yan. 2019. Time series prediction method based on Convolutional Autoencoder and LSTM. In *2019 Chinese Automation Congress (CAC)*. IEEE, Hangzhou, China, 5790–5793. https://doi.org/10.1109/CAC48633.2019.8996842

[59] Zhong Zheng and Zijun Zhang. 2023. A temporal convolutional recurrent autoencoder based framework for compressing time series data. *Applied Soft Computing* 147 (2023), 110797. https://doi.org/10.1016/j.asoc.2023.110797