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

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

Publication date:
2024

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

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 time series data that is ingested on the edge before it is transferred to the cloud where it is stored and queried. This results in at least four challenges: 1) High frequency time series data must be ingested on limited hardware fast enough to keep up with the generation; 2) Limited bandwidth makes it impossible to transfer the data without compression; 3) High storage costs when data is stored; and 4) Low data quality to unbounded lossy compression methods commonly used by practitioners. Practitioners currently use solutions that only solve some of these challenges. In this paper, we evaluate a solution for the entire pipeline based on the Time Series Management System ModelarDB that addresses all four challenges efficiently. With ModelarDB, the user can exploit both lossless and error-bounded lossy compression. We evaluate the solution in a realistic edge-to-cloud scenario with real-world data under different aspects. For lossless compression, ModelarDB achieves up to 1.5x better compression and 1.2x better transfer efficiency than lossless solutions commonly used by practitioners. For lossy compression, ModelarDB offers significant compression comparable to a lossy method commonly used by practitioners today. However, ModelarDB has orders of magnitude smaller errors.

ACM Reference Format:

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

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

1 INTRODUCTION

Wind turbines are monitored by many high-quality sensors that generate massive amounts of time series at high frequencies such as 10 Hz, 50 Hz or 100 Hz. As an 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 [26]. The data is collected at the edge and transferred to the cloud for large scale analysis as shown in Figure 1. However, the edge nodes have hardware similar to low-end commodity PCs and the available bandwidth is limited. This creates the following challenges:

Challenge 1: High Frequency Data. Data ingestion on the edge must be fast enough to keep up with data generation on limited hardware, e.g., without a GPU. Thus, use of slow or resource intensive techniques like autoencoders is not possible [9, 34, 41, 52].

Challenge 2: Limited Bandwidth. For Renewable Energy Sources (RES) installations, the network bandwidth between edge and cloud can be as low as 0.5 Mbit/s [26, 27]. Thus, it is impossible to transfer the raw high frequency time series without significant compression.

Challenge 3: High Storage Cost. Storage of the raw time series in the cloud leads to prohibitively high storage costs, e.g., storing one year of data for a turbine that generates 321 GiB daily on Amazon S3 costs ~1851$ while it costs ~925500$ for a park of 50 turbines [2].

Challenge 4: Low Data Quality After Compression. To reduce bandwidth and storage usage, lossy compression without error bounds is often used. However, such methods do not provide data quality guarantees and thus may impact the result of analytics [45].

Practitioners currently use solutions (LLC and AGG in Table 1) that only solve a subset of these challenges, these solutions are:
Table 1: Challenges solved by each of the solutions.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Challenge 1</th>
<th>Challenge 2</th>
<th>Challenge 3</th>
<th>Challenge 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLC</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>AGG</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>MDB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Lossless Compression through Big Data Formats (LLC). The edge nodes store the time series using well-known big data formats with lossless compression like Apache Parquet or Apache ORC and these files are transferred to the cloud (LLC in Figure 1). This provides fast ingestion and preserves data quality, but lossless compression provides very low compression factors [31] as shown in Table 1. Unbounded Lossy Compression through Simple Aggregates (AGG). The edge nodes compute simple aggregates for a static time interval, e.g., 10 minutes, using an aggregation function such as mean (shown as AGG in Figure 1). This provides fast ingestion and a high compression factor, but results in low data quality due to the use of unbounded lossy compression as shown in Table 1.

From Table 1 it is clear that neither of these solutions solves all four of the challenges, thus practitioners currently have to make a trade-off between bandwidth and storage usage and data quality. To provide a solution that solves all four challenges, the Time Series Management System (TSMS) ModelarDB (MDB) [24, 25, 27] was proposed. MDB is developed in on-going research and has already been pilot deployed [1, 45]. MDB efficiently manages RES data through lossless and error-bounded lossy compression through the use of models from which the original values can be reconstructed within a user-defined per-value relative error bound (possibly 0%). MDB’s efficiency comes from two key insights about RES data. First, for some time series it is acceptable to trade a small amount of error for significantly better compression while for other it is not. Thus, MDB supports both lossless and error-bounded lossy compression. Second, the time series have long subsequences that can be efficiently represented using simple functions, i.e., 1st and 2nd degree polynomials, but some subsequences cannot. Thus, MDB implements multiple different lossless and lossy compression methods and automatically uses the method that provides the best compression factor for each dynamically sized subsequence.

MDB is designed to be modular and uses different query engines and data stores depending on its configuration. Thus, it can use a lightweight efficient query engine and data store on the edge and a scalable query engine and data store in the cloud. MDB provides a complete data management pipeline across edge and cloud and is shown as MDB in Figure 1. On the edge, it compresses data points from the sensors efficiently into segments containing metadata and models. These segments are automatically transferred to the cloud, thus significantly reducing the amount of storage required. When the segments have been transferred to the cloud, clients can query the data points using SQL with the knowledge that MDB guarantees that all data points are within the relative error bound. A much more detailed description of MDB is provided in Section 3.

In this paper, we provide insights into how MDB performs in an edge-to-cloud distributed environment using real-life datasets provided by our partners in the RES industry. As part of our evaluation we compare MDB to the solutions currently used in the RES industry. The evaluation is based on our decade-long experience with RES data and our collaboration with industry partners in the MORE project [1] and with a major wind turbine manufacturer. We formulate three research questions (RQs) and related subquestions:

**RQ1**: How does a high frequency wind turbine dataset compress with the evaluated solutions? **RQ1.1**: How does MDB compare against LLC in terms of compression factor? **RQ1.2**: How does MDB compare against AGG 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 does MDB’s model types are used? **RQ2.1**: What is the distribution of model types? **RQ2.2**: What is the length of segments for each model type?

**RQ3**: How is the transfer efficiency of the three solutions?

**RQ4**: How well does MDB preserve the data quality of a high frequency wind turbine dataset? **RQ4.1**: What is the compression error of a high frequency wind turbine dataset when compressed using MDB? **RQ4.2**: How much of the high frequency wind turbine dataset is compressed without any error using MDB?

In summary, we make the following contributions in this paper:

1. We evaluate MDB as a solution for the practical problem of efficiently managing high frequency wind turbine data across edge and cloud with a focus on storage usage, bandwidth usage, and data quality using real-life data and report key insights;
2. We compare MDB to the solutions currently used in the RES industry;
3. We provide a framework with source code for evaluating the impact on the three solutions when varying three orthogonal aspects.

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

ModelarDB (MDB) [24, 25, 27] is a Time Series Management System that can be deployed on edge nodes and cloud nodes and, which automatically transfers compressed time series data from the edge to the cloud. MDB was designed to be modular and consists of a Java core which can be interfaced with different query engines and data stores. This allows the same MDB binary to be used on the edge and in the cloud. Which query engine and data store to use is specified using a configuration file. For example, on the edge MDB can use the lightweight embeddable RDBMS H2 as its query engine and data store, and in the cloud Apache Spark and Apache Cassandra can be used as the query engine and data store, respectively.

We use H2 as query engine and data store for the edge. The cloud node uses Apache Spark for query processing and Apache ORC files written to a local file system as a data store. To measure MDB’s compression, we measure the size of its Apache ORC files on the cloud. Apache ORC is also the format for the other solutions.

MDB’s architecture consists of three sets of components and it is shown in Figure 2 with a configuration suitable for the edge, data transfer, and a configuration suitable for the cloud.

Data Ingestion. These components ingest data points and compress them using models that are functional approximations for dynamically sized subsequences. Modified versions of Poor Man’s Compression-Mean (PMC) [31], Swing Filter (Swing) [13] and the lossless compression method for floating-point values proposed for the Gorilla TSM5 (Gorilla) [39] are used as model types. The model types are used for constructing models from time series. PMC and Swing are piecewise constant and linear approximation methods, respectively, while Gorilla is a sequential method that compresses values using an XOR-based encoding. The model types used by MDB are computationally efficient allowing for a fast ingestion which addresses Challenge 1 (Slow ingestion). For each model type, MDB fits a value at a time to a model until the error bound is exceeded. The model with the best compression is stored to represent the subsequence. Given that Gorilla is a lossless compression method, it can compress any number of values in a single model and a user-configurable length bound is thus used to ensure finite length segments. The default length bound of 50 values is used. In Figure 3, MDB fits a Swing model to one value at a time until the error bound is exceeded. Then the segment is created from the model and other metadata (segment $S$ in Figure 3). Segments are used for transferring and storing time series. MDB’s use of a relative user-defined pointwise error bound addresses Challenge 4 (Low data quality). A study of MDB’s base model types’ (i.e., PMC and Swing) impact on the time series forecasting accuracy, shows that the compression using error bounds up to 23% and 15% for PMC and Swing, respectively, has a very minor impact on the accuracy of forecasting methods [10]. The results also show that in some cases, the accuracy is even improved. Thus, MDB’s lossy compression goes hand in hand with good downstream accuracy.

Query Processing. These components perform caching of segments and execution of queries on locally stored segments. Thus, query processing is supported by both the edge and cloud nodes. Large analytical queries are handled on the cloud since segments from all edge nodes are consolidated on the cloud. MDB’s model-based compression allows common aggregate queries to be computed directly from the models and metadata improving the query performance by up to 573x compared to popular systems and Apache ORC and Apache Parquet file formats [25].

Segment Storage. These components enable reading from and writing to local data stores and transfer of segments to a remote data store, named Remote in Figure 2, which is another instance of MDB running on the cloud. Apache Arrow Flight is used for the communication and transferring of segments between MDB instances. Transferring and storing segments instead of data points allow MDB to efficiently handle Challenges 2 (Limited bandwidth) and 3 (High storage costs), respectively. In our evaluations, MDB edge transfers segments to the remote data store i.e., the cloud node where all segments are consolidated.

EVALUATION SETUP

4.1 Currently Used Solutions

LLC. The high frequency time series are written to big data files with Snappy compression (default) by Apache Arrow v11.0.0 [15]. AGG. The high frequency time series are aggregated by a fixed time interval and the simple aggregates are stored in big data files like for LLC. Specifically, we use the common method of splitting the time series into fixed time periods and computing the mean [36]. Section 4.2.3 describes the SIs (i.e., time periods) used for AGG. We use SIs that are commonly used by our industry partners.

4.2 Evaluation Aspects

4.2.1 Dataset. We use real-life datasets from multiple wind turbines and wind park controllers provided by our industry partners. Table 2 provides a summary of the three real-life datasets. We cannot disclose and share the first two datasets named Power Controller...
Dataset (PCD) and Multiple Turbines Dataset (MTD) as they are business secrets of our industry partners. Evaluation results can be reproduced on the third and public real-life dataset named Wind Turbine Measurements (WTM) [11].

Each dataset is stored in a single file (i.e., one file per dataset) using a big data format compressed with the Snappy compression algorithm. The datasets are sorted by the id and timestamp.

PCD. The dataset was collected by the wind park power controller from a single wind park for ≈2 years and 4 months. It is a multivariate time series with 150ms SI. The dataset has 10 signals of 32-bit floating values and consists of ≈480M data points. The majority of PCD’s time series are regular, but in some cases, the time between consecutive data points deviates slightly from 150ms, so we applied a minor preprocessing step to produce regular time series, which is a requirement of the MDB implementation [37] we are using.

MTD. The dataset contains wind turbine measurements from several wind parks. Every wind turbine collects 10 signals with 2s SI. 4 out of 10 signals are cosine and sine transformations of other signals (i.e., derived signals) that are already present in the dataset (i.e., source signals). The results for the derived signals closely match the results of the respective source signals meaning that they had the same compression factor as well as similar use of models. Thus, we removed the derived signals and only use source signals. The mean period of recording among all wind turbines is 11 months.

WTM. This is an existing real-life dataset published as part of [10]. It originates from the same source as MTD and it contains measurements from a single wind turbine sampled every 2 seconds for 10 days. The time series has 10 signals in 32-bit floating values and consists of ≈300K data points. We can see that its compression is lossless. Figure 4 shows the file sizes of the compressed datasets for MDB, Apache ORC and Apache Parquet formats. We can see that Apache ORC compresses better than Apache Parquet for all datasets which match previous work [24, 25].

### 4.3 Metrics

#### 4.3.1 Compression Effectiveness

Compression Factor (CF) is used to measure the effectiveness of each compression method. The metric computes the ratio between the size $s$ of the original dataset and the size $s'$ of the compressed dataset as $CF = s / s'$.

#### 4.3.2 Transfer Efficiency

We compute the number of values transferred from the edge node to the cloud node per second.

#### 4.3.3 Data Quality

To measure the quality of reconstructed datasets, we use two metrics: $MAPE$ (Mean Absolute Percentage Error) and $MAPE_\epsilon$. $MAPE_\epsilon$ is an extension of $MAPE$ that is computed only for lossy compressed data points where $e_i$ is not equal to 0.

\[
MAPE_\epsilon = \frac{1}{\sum_{s \in S} S} \sum_{s \in S} e_i
\]

where $n$ is the number of all data points in the dataset and $e_i$ is the pointwise error at $i$.

#### 4.4 Configuration

To evaluate the solutions, we setup a RES data management infrastructure with a single edge node and cloud node. An edge node has 2 CPU Cores with 4 threads, 4 GiB RAM and an HDD to match the edge node configuration in [26]. The cloud node has 16 CPU Cores, 256 GiB RAM and 8 SSD disks. Both nodes run on Ubuntu 20.04 LTS.

For testing the transfer efficiency of MDB and the other solutions, we reduce the network bandwidth to 512Kbit/s and 2560Kbit/s to emulate the low bandwidth scenario in [26].

### 5 RESULTS ANALYSIS

#### 5.1 Compression Factor

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

##### 5.1.1 MDB against LLC

As LLC compresses only losslessly, we first compare it with MDB at $\epsilon=0\%$ meaning that MDB guarantees that its compression is lossless. Figure 4 shows the file sizes of the compressed datasets for MDB, Apache ORC and Apache Parquet formats. We can see that Apache ORC compresses better than Apache Parquet for all datasets which match previous work [24, 25]. The results in [17, 24, 25] show that both formats have very similar compression times. Thus, only Apache ORC is used below. The results also show that MDB provides 1.5x, 1.4x and 1.3x better lossless compression than Apache ORC for PCD, MTD and WTM, respectively.

In Figures 5a and 5b, we evaluate MDB’s improvement in CF over LLC when the $\epsilon$ is above 0%. Discussions with industry partners revealed that having sufficient data quality is a primary requirement when choosing between solutions and a small pointwise error bound can be tolerated. Thus, we compare MDB’s lossy compression to LLC to get more insights into the benefits of MDB. Figures 5a-5b display improvement in MDB’s CF over LLC’s CF for all datasets for different error bounds. In Figure 5a, the $y$ axis uses a log scale to better display the impact of introducing very low $\epsilon$ and Figure 5b uses a linear scale for the rest of the datasets, while $x$ axis
MTM shows file sizes of PCD datasets under different SIs compressed 1.5x to 2.1x between 0.5% and 10% error than AGG with the same CF. In contrast to MDB, AGG results in Figures 6d-6f show that MDB introduces much smaller error than the 7x aggregation. MDB at \( \varepsilon = 1\% \) for MTD and WTM at \( \varepsilon = 10\% \) provides comparable CFs to 3x aggregation, respectively. To match the CF of aggregation with higher SIs, MDB requires \( \varepsilon > 10\% \). Having more than 10% error might seem to result in a significant loss of data quality, however, the results in Figures 6d-6f show that MDB introduces much smaller error than AGG with the same CF. In contrast to MDB, AGG does not provide any data quality guarantees thus leading to a significant reduction of the quality of compressed values. To measure data quality, we compute MAPE and maximum pointwise error (MPE) for the whole dataset. In Figure 6e, MDB with \( \varepsilon = 60\% \) and 99% provides comparable CF as 15x (30s) and 30x (1m) AGG for MTD. However, 15x and 30x AGG for MTD provides 1.5e+8 and 1.4e+8 MAPEs that are larger than MDB’s respective error bounds by \( \sim 8 \) orders of magnitude, while MPE is \( \sim 17 \) orders of magnitude higher. For WTM, MDB at \( \varepsilon = 40\% \) and 99% matches the CF of 5x (10s) and 15x (30s) AGG, whereas AGG’s MAPEs are \( \sim 3 \) orders of magnitude higher. We do not compute MDB with \( \varepsilon = 100\% \) as that allows any value to become zero. When it comes to preserving data quality, we can see that for all datasets, MAPE for AGG is 3-8 orders of magnitude higher than for MDB, while the MPE is 9-17 orders of magnitude higher. For example, in Figure 6d, the MPE for PCD is equal to 1.64e+35 after aggregating with SI=10m. As a result of aggregation, the original value \( v_i = 1.18e-38 \) was aggregated into the value \( v_i' = 0.001934 \). In addition to very high error, AGG generates undefined errors when \( v_i = 0 \) due to division by zero. We excluded them to compute MAPE and MPE. In contrast, to maintain the pointwise \( \varepsilon \), MDB stores \( v_i = 0 \) without any error. The results show that AGG removes informative outliers and fluctuations in a dataset that might be critical to certain analytical tasks. We discuss MDB’s compression error in detail in Section 5.4.

From the findings in this section, we answer RQ1.1 as follows. For lossless compression, MDB outperforms LLC and the results showed that allowing a very small \( \varepsilon \) 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 by MDB significantly better than the other datasets.

### 5.1.2 MDB Compared to AGG

In this section, we compare MDB against AGG to determine the tradeoff of using each method in terms of compression and data quality. We use the error bounds in Section 4.2.2 for MDB. We perform AGG as described in Section 4.1 and use SIs that are commonly used by our industry partners as described in Section 4.2.3.

Figures 6a-6c show CF for MDB and AGG for each dataset. Table 4 shows file sizes of PCD datasets under different SIs compressed by LLC and MDB with \( \varepsilon = 0\%, 0.1\%, 1\% \) and 10%.

Table 4: File sizes of aggregated PCD datasets in MB

<table>
<thead>
<tr>
<th>SI</th>
<th>ORC</th>
<th>( \varepsilon = 0% )</th>
<th>( \varepsilon = 0.1% )</th>
<th>( \varepsilon = 1% )</th>
<th>( \varepsilon = 10% )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15s</td>
<td>13755</td>
<td>8901</td>
<td>4333</td>
<td>1617</td>
<td>278</td>
</tr>
<tr>
<td>1.05s</td>
<td>2089</td>
<td>1462</td>
<td>1004</td>
<td>413</td>
<td>58</td>
</tr>
<tr>
<td>2.1s</td>
<td>1063</td>
<td>755</td>
<td>571</td>
<td>279</td>
<td>37</td>
</tr>
<tr>
<td>4.95s</td>
<td>436</td>
<td>331</td>
<td>269</td>
<td>173</td>
<td>23</td>
</tr>
<tr>
<td>10.05s</td>
<td>231</td>
<td>167</td>
<td>140</td>
<td>102</td>
<td>16</td>
</tr>
<tr>
<td>1m</td>
<td>39</td>
<td>30</td>
<td>27</td>
<td>22</td>
<td>7.1</td>
</tr>
<tr>
<td>10m</td>
<td>5</td>
<td>3.3</td>
<td>3.2</td>
<td>2.8</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Figures 6a-6c show the CF of MDB and WTM only increases from 1.7x to 3.3x and from 29.6x and 49x at \( \varepsilon = 0.5\% \), \( \varepsilon = 1\% \), \( \varepsilon = 5\% \) and \( \varepsilon = 10\% \), whereas in Figure 5b, CF of MTD and WTM only increases from 1.7x to 3.3x and from 1.5x to 2.1x between 0.5% and 10% \( \varepsilon \), respectively. To conclude, we answer RQ1.1 as follows. For lossless compression, MDB outperforms LLC and the results showed that allowing a very small \( \varepsilon \) 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 by MDB significantly better than the other datasets.
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 $\epsilon = 0\%$ (i.e., lossless compression) provides slightly lower compression than LLC due to the very small size of the dataset where the amount of MDB’s segment metadata becomes overhead. Overall, the results for WTM shown in Figure 7c are very similar to MTD.

From these results, we can answer RQ1.3 as follows. MDB provides the best CF for datasets with a very short SI such as PCD. For datasets with higher SI where the variability between the values is very high, we can see that MDB still compresses better than LLC. However, MDB’s compression becomes less effective and the impact of the $\epsilon$ on CF decreases when the SI of the dataset is very high. The results also show that the SI of a dataset almost has no impact on the LLC’s CF.

### 5.2 Usage of MDB model types

In this section, we analyze the use of each model type used for MDB. Specifically, the ratio of compressed values and the length of segments created by each model type are analyzed.

#### 5.2.1 Distribution of model types

Figures 8a-8c show the ratio of compressed values represented by each model type when compressing each dataset. 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 $\epsilon = 0\%$ for MTD with SI=10m, where data volumes are small. As an example, for PCD MDB at $\epsilon = 0\%$ (lossless compression) 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 $\epsilon = 0\%$ compresses better than LLC due to the very small size of the dataset where the amount of MDB’s segment metadata becomes overhead. Overall, the results for WTM shown in Figure 7c are very similar to MTD.

From these results, we can answer RQ1.3 as follows. MDB provides the best CF for datasets with a very short SI such as PCD. For datasets with higher SI where the variability between the values is very high, we can see that MDB still compresses better than LLC. However, MDB’s compression becomes less effective and the impact of the $\epsilon$ on CF decreases when the SI of the dataset is very high. The results also show that the SI of a dataset almost has no impact on the LLC’s CF.
and the other datasets. Gorilla is the most used model type for compressing at $\varepsilon=0\%$ and $\varepsilon=0.01\%$ among all datasets. At $\varepsilon=0\%$, it represents 83.8%, 72.9% and 62.7% values for PCD, MTD and WTM, respectively, while the rest of the values are mainly compressed by PMC. Swing is only used for compressing 1% of MTD at $\varepsilon=0\%$. Allowing $\varepsilon=0.01\%$ enables significant use of Swing for all datasets. This is also aligned with the results in Section 5.1.1 where we can see significant increase in CF after allowing minor error. As we increase the $\varepsilon$, the use of Gorilla is significantly decreasing at the expense of other model types. Particularly, at $\varepsilon=0.1\%$ Swing represents 36.7%, while PMC represents 30.4% of PCD’s values. At $\varepsilon=1\%$, PMC represents half of PCD’s values, while at $\varepsilon=10\%$ its ratio increases to 65.2% followed by Swing with 33.7%. Figures 8a-8c also show that increasing $\varepsilon$ leads to an increased use of Swing over other model types for MTD dataset. Its ratio increases from 15.7% at $\varepsilon=0.01\%$ to 30.2% at $\varepsilon=5\%$ and 37.9% at $\varepsilon=10\%$. Similarly, Swing has the highest increase rate from 8.0% at $\varepsilon=0.01\%$ to 30.6% at $\varepsilon=10\%$ for WTM. It should also be noted that Gorilla is still required to compress 24.6% of MTD and 33.9% of WTM at $\varepsilon=10\%$, while all of PCD is compressed by PMC and Swing.

To conclude, we answer RQ2.1 as follows. The distribution of model types depends on the dataset, its SI and $\varepsilon$ when compressing it. When $\varepsilon=0\%$, Gorilla is the most used model type for all datasets. However, there are still a significant amount of constant subsequences represented by PMC in all datasets when we compress at $\varepsilon=0\%$. When $\varepsilon=0\%$, a dataset with a very low SI lends itself better to the model types that provide lossy compression than a dataset with a higher SI. When using lossy compression, most of PCD’s values are represented by PMC, whereas datasets with higher SI are more represented with Swing as we increase $\varepsilon$. Gorilla is being used to compress significant amounts of values in datasets with higher SI even when compressing with $\varepsilon=10\%$. This points to the compression effectiveness of MDB’s multi-model compression.

5.2.2 Segment length. Figures 8d-8f show the mean length of segments created by each model type when compressing each dataset. We can see that on average PMC created the longest segments for all datasets under all $\varepsilon$ with the exception of $\varepsilon=5\%$ and $\varepsilon=10\%$ in PCD. At $\varepsilon=0\%$, PMC has the longest mean segment length for all datasets. Thus, we can conclude that there are some long subsequences within all three datasets that can be represented by a constant value. As we allow for some $\varepsilon$, the mean length of PMC segments decreases for all datasets. Figures 8a-8c showed that the amount of values represented by PMC increases with $\varepsilon$. Compressing with some error allows more PMC segments to be created with less values than compressing at $\varepsilon=0\%$. Figures 8g-8i show the median length of segments for each model type. Computing the median length helps to minimize the impact of outliers (i.e. extremely long segments) on the analysis. We can see that the median length of PMC segments is decreasing as we increase $\varepsilon$. For example, the mean length of PMC segment increases by 3.9x between $\varepsilon=0.2\%$ and $\varepsilon=1\%$ for PCD. On the contrary, the median PCD segment length decreases from 16 values to 6 values between $\varepsilon=0.2\%$ and $\varepsilon=1\%$. We can see that PMC effectively approximates both very long and short segments...
in PCD. A similar tendency can be observed for Swing in MTD and WTM. The mean length of its segments increased from 49 to 90 values between $\epsilon=0.01\%$ and $\epsilon=10\%$ in Figure 8e for MTD, while in Figure 8g, the median length decreased from 30 to 24. Both Figures 8d and 8i show that Gorilla segments are all below 50 values due to the length bound parameter that we defined for Gorilla before compressing datasets. It should be noted that Gorilla’s length bound parameter was optimized for current datasets and $\epsilon$. In Figure 8g, we can see that at $\epsilon=0\%$ and $\epsilon=10\%$ PMC reaches the lowest median segment length with 5 and 6 values. However, having a PMC segment of 5 values is more desirable than Gorilla segment of 5 values as Gorilla can compress 5 values with at least 36 bits, while PMC always uses 32 bits per segment. In addition, results in Figures 8d-8i show that there is a clear distinction between PCD and the other datasets. This is explained by the difference in SI which also affects the length of segments created by PMC and Swing. In addition, computing measures of dispersion for all datasets showed that there is less variability in PCD values compared to other datasets.

To conclude, we can say that dataset’s SI and MDB’s $\epsilon$ affect the length of segments. We can see that model types providing lossy compression are more effective than the lossless model type for the very high-frequency dataset. This can be seen in the increasing mean length and decreasing median length of created segments. This means PMC and Swing are effectively approximating subsequences with very few values as well as hundreds of values. The majority of PMC segments for PCD are very short consisting of up to 20 values, but there are also segments that store hundreds of values as can be seen from Figure 8d. An increase in $\epsilon$ leads to an increase in the mean length and decrease in the median segment length, meaning that PMC is approximating more patterns. This can also be confirmed from the findings in Section 5.2.1 which showed that PMC was the most effective model type for representing PCD. We see a similar pattern with Swing for MTD and WTM. Particularly, the most observed Swing segments are 19-30 values long and the mean length is increasing from 20-30 values at $\epsilon=0\%$ to ~90 values at $\epsilon=10\%$. This is also aligned with the observation in Section 5.2.1 where compression performance of Swing increases more than other model types as we increase $\epsilon$ in MTD and WTM. The lossless Gorilla model type is the most used model type for compressing at very low $\epsilon$ given that it always stores values with zero error. We also observed that the second most observed segment length for Gorilla was one value meaning that it is used heavily for representing outliers. In ModelarDB’s implementation of MDB, Gorilla is used to store outliers with one value, however, PMC can also be used instead given that it also represents subsequences of one value using 32 bits.

To conclude, we can answer RQ2.2 as follows. The segment lengths for each model type depend on SI, $\epsilon$, and dataset. In general, PMC has higher mean length than the other model types. The maximum length of a PMC segment goes above 1000 values at 0% $\epsilon$. In almost all cases, Swing creates segments with shorter mean length than PMC, and its median segment length is closer to the mean length. Both the mean and median length of Gorilla segments are always similar due to the length bound parameter. As we increase $\epsilon$, model types with increasing mean and decreasing median segment length represent more values and thus achieve better compression performance. This means that the model type

![Figure 9: Values ingested and transferred per second for PCD.](image)

is representing higher variety of subsequences. We observe this tendency with PMC on PCD and Swing on MTD and WTM.

### 5.3 Transfer Efficiency

The limited bandwidth (Challenge 2) from RES installations can hinder the transfer of data from edge to cloud. In this section, we evaluate how many values can be transferred from edge to cloud by MDB, LLC and AGG when the bandwidth is 512Kbit/s. The challenge is more exacerbated for more high frequent data and thus we here focus exclusively on PCD. We perform the experiment with 2 days of data. The environment is emulated as described in Section 4.4. MDB on the edge and the cloud is configured as described in Section 3. For AGG, we use SI=1.05s (i.e., aggregating 7 values into 1) that achieves the lowest compression error for AGG. For LLC and AGG, we implemented a Java program for ingesting data points as described in Section 4.1 and its results are stored in Apache ORC format, which provided better compression than Apache Parquet (Section 5.1.1). Transfer to the cloud is then performed using scp. For all three 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 9(a), MDB ingests from 1.3 to 3.1 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 as shown next.

Figure 9(b) shows the number of values handled per second when the ingested data is also transferred from edge to cloud. With LLC around 19000 values are transferred per second. For $\epsilon=0\%$, MDB transfers 1.2x faster than LLC. As $\epsilon$ is increased, MDB transfers more values per second and this is correlated with the increase in CF shown in Section 5.1.1. AGG transfers 1.05x more values per second than MDB with $\epsilon=1\%$. As shown in Section 5.1.2, AGG, however, produces unbounded errors that are many orders of magnitude higher than MDB and LLC thus fails in terms of Challenge 4. With $\epsilon=5\%$ MDB transfers 7.2e+5 values per second, i.e., 38x and 6x more than LLC and 7x AGG, respectively, and with $\epsilon=10\%$ MDB transfers 1.2e+6 values per second, i.e., 64x and 10x more than LLC and 7x AGG, respectively (not shown in Figure 9(b)). With a bandwidth of 25Mbit we saw similar, but ~ 5x faster results.

To conclude, we answer RQ3 as follows. LLC, can transfer around 19,000 values per second with a 512 Kbit/s connection, while MDB at $\epsilon=0\%$ (i.e., with lossless compression) can transfer 1.2x more values. In addition, we saw that MDB can transfer even more values through the use of error-bounded lossy compression. As an example, MDB’s error-bounded lossy compression allows transfer of 6x more values.
than LLC at only $\epsilon = 1\%$ with 512 Kbit/s. MDB’s error-bounded lossy compression with $\epsilon = 1\%$ matches the transfer efficiency of 7x AGG while also having up to ~12 orders of magnitude better data quality. Our results also showed the overhead of ingestion is insignificant compared to the available network bandwidth on the edge.

### 5.4 Data quality

Discussions with industry partners revealed that having data quality guarantee is one of the primary requirements when choosing between the solutions. Section 5.1.1 showed that AGG has extremely high errors and removes informative outliers and fluctuations. On the contrary, MDB uses a pointwise $\epsilon$ meaning that the compression error $\epsilon$ is guaranteed to be $0 \leq \epsilon \leq \epsilon$. In this section, we measure the compression error of MDB using MAPE and MAPE$\epsilon$. Then we measure the percentage of values represented with zero error.

#### 5.4.1 Compression Error

Figure 10 shows the distribution of MAPE and MAPE$\epsilon$ for all signals of each dataset. Whiskers of box plots represent the minimum and maximum values. Whiskers that span all the way down the x-axis represent zero. In addition to MAPE, we compute MAPE$\epsilon$, which gives additional insight into the degree of error when $\epsilon > 0$. MAPE$\epsilon$ is higher than MAPE by 1.02x-99154x for PCD, 1.002x-282x for MTD and 1.003x-6750x for WTM. The big difference between MAPE, and MAPE$\epsilon$ is due to the large number of losslessly compressed values in some signals, which significantly decreases MAPE. The results also show that both MAPE and MAPE$\epsilon$, are at least 0.54x and 0.38x lower than $\epsilon$ for all datasets. PCD in comparison to the other datasets, shows a higher MAPE. This is related to the higher use of PM and Swing to compress PCD for all $\epsilon$ as shown in Figures 8a-8c. In addition, Figures 8a-8c also show that Gorilla is significantly used for compressing MTD and WTM even when we allow for some error. This explains the lower MAPE for MTD and WTM. At $\epsilon = 0.01\%$, the highest MAPE and MAPE$\epsilon$ for PCD is 4.7e-3 and 5.2e-3, respectively. At the same $\epsilon$, MTD’s and WTM’s maximum MAPEs are 1.1e-3 and 1.5e-3 MAPE, while their MAPE$\epsilon$s are 6.1e-3 and 3.1e-3, respectively. We can see that for all datasets the highest MAPE is equal to 0.47x of $\epsilon$. PCD’s lowest MAPE and MAPE$\epsilon$ are equal to 0 for all $\epsilon$ meaning that a particular signal PowerLowerLimit is represented losslessly by MDB’s PM model type for all $\epsilon$. This also shows the effectiveness of MDB with constant signals. As we increase $\epsilon$, we can see a further decrease of PCD’s MAPE in relation to $\epsilon$. At $\epsilon = 0.1\%$, PCD’s highest MAPE is 0.42x of $\epsilon$, while at $\epsilon = 1\%$ and 10%, it decreases to 0.37x and 0.29x, respectively. Similarly, PCD’s highest MAPE$\epsilon$, is 0.45x, 0.38x and 0.3x of $\epsilon$ at 0.1%, 1% and 10% $\epsilon$, respectively. Contrary to PCD, the difference between $\epsilon$ and compression error gradually decreases for the other datasets. As an example, the highest MAPE for MTD increases from 0.27x of $\epsilon$ at $\epsilon = 0.1\%$ to 0.34x of $\epsilon$ at $\epsilon = 10\%$. The results for WTM are very similar as it is a subset of MTD. We can see two different trajectories in the growth of compression error for the PCD and the other datasets. The increase in the compression error for MTD and WTM is due to the higher use of Swing model type at high error bounds, while the decrease in the compression error for PCD is explained by the high use of PM.

To conclude, we answer RQ4.1 as follows. MDB’s compression error on a high-frequency wind turbine dataset is significantly lower than $\epsilon$. The highest MAPE$\epsilon$ for an individual signal of a dataset reaches 0.47x, 0.36x and 0.38x of $\epsilon$, while the highest MAPE$\epsilon$ for a signal reaches 0.52x, 0.61x and 0.44x of $\epsilon$ for PCD, MTD and WTM, respectively. Signals with constant or slowly changing values are compressed with zero error. When compressing the very high-frequency dataset such as PCD, MAPE and MAPE$\epsilon$, further decrease in relation to $\epsilon$. This means that compressing with higher $\epsilon$ values, the compression error from MDB is significantly lower than $\epsilon$. This can be explained by decreasing median length of PMC and Swing segments as it leads to a representation of fewer but more accurate values in each segment. As discussed in Section 5.2.2, the median length of PMC and Swing segments decreases as $\epsilon$ increases.

#### 5.4.2 Losslessly Compressed Data Points

Figure 11 shows the ratio of values that MDB represents losslessly after compression. The results show that more than half of the compressed values are identical to their original values among all datasets when $\epsilon \leq 0.05\%$. For all $\epsilon$, PCD has the least amount of exactly represented values compared to the other datasets. This is related to the higher use of PM and Swing as they allow for lossy compression. 71.9% and 53.2% of PCD values are compressed with zero error at $\epsilon = 0.01\%$ and 0.05%, respectively, whereas MTD and WTM have 89.1%, 79.6% and 95.0%, 85.3% exactly represented values, respectively. We can see further gradual decrease in the amount of losslessly represented values for all datasets as $\epsilon$ increases. At $\epsilon = 1\%$, more than 60% of the compressed MTD and WTM values are identical to the original values, while PCD has only 25.2% values compressed with zero error. We can see that at $\epsilon = 10\%$, PCD, MTD and WTM have 71.1%, 31.9% and 34.3% of values compressed with zero error. This shows that the significant use of Gorilla for compressing MTD and WTM at $\epsilon = 10\%$ as can be seen in Figures 8a-8c, increases the amount of values represented with zero error. As the large majority of PCD is represented with PM and Swing at $\epsilon = 10\%$ as can be seen in Figures 8a-8c, we can conclude from Figure 11 that PM and Swing also losslessly represent significant amount of values ($17\%$). This is partly due to the signal PowerLowerLimit being losslessly represented by PM. Also certain periods when park operation is halted and thus 0 power is being produced lead to the generation of constant values for most signals. In addition, the decreasing median length of both PM and Swing segments at higher $\epsilon$ values (Figures 8g-8i) also increases the probability of representing values losslessly.

To conclude, we answer RQ4.2 as follows. MDB represents 71.9%-95.0% of values without any error at $\epsilon = 0.01\%$. More than half of the values of all datasets are represented losslessly at $\epsilon = 0.05\%$. More than 60% of the values of MTD and WTM are represented without error at $\epsilon = 1\%$. In general, the ratio of losslessly represented values depends on which model types and $\epsilon$ are used. PMC and Swing represent up to 17.1% of the values losslessly of a very high-frequency (i.e. PCD) dataset at $\epsilon = 10\%$.

### 6 RELATED WORK

#### 6.1 Time series compression

Time series compression is a broad field and a plethora of methods have been developed. A survey on time series compression techniques [9] divides them into five categories: dictionary-based, functional approximation, autoencoders, sequential methods and others. Further, the survey uses the following combination of classes to
These methods are generally adaptive and non-symmetric, either lossy or lossless. TRISTAN [35], CORAD [30], Accelerometer LZSS atoms to represent sequences of time series using an atom key. Methods utilize different operations to compress and decompress in reverse order to decompress time series, while non-symmetric compressed. Symmetric methods perform the same operations but return identical data when the compressed representation is decompressed using a lossy method, while lossless methods always guaranteed to return identical values to the original time series if it is compressed. Decompression of compressed time series is not guaranteed by a training phase on a given dataset, while adaptive methods do. Decompression of compressed time series is not guaranteed to return identical values to the original time series if it is compressed using a lossy method, while lossless methods always return identical data when the compressed representation is decompressed. Symmetric methods perform the same operations but in reverse order to decompress time series, while non-symmetric methods utilize different operations to compress and decompress time series.

Dictionary-based compression techniques split time series into equal-length subsequences called atoms and build a dictionary of atoms to represent sequences of time series using an atom key. These methods are generally adaptive and non-symmetric, either lossy or lossless. TRISTAN [35], CORAD [30], Accelerometer LZSS [40] and Differential LZW [32] are examples of dictionary-based compression techniques. Maximizing the search speed within the dictionary and creating segments are the main challenges of these techniques when applied for online compression. Functional approximation techniques represent time series using a function of time. Given that in the majority of cases, it is infeasible to represent the whole time series using a single function, the time series is split into segments and each segment is approximated by a single function. To minimize the search space of functions, implementations consider a set or a family of functions. Piecewise Polynomial Approximation (PPA) [12], Chebyshev Polynomial Transform [20], Discrete Wavelet Transform, Discrete Fourier Transform [28], Discrete Cosine Transform [21] and DCTZ [50] are examples of functional approximation techniques. These techniques are non-adaptive meaning that no prior training phase is required.

Autoencoders are a particular family of neural networks that consist of two parts: an encoder that receives the time series and returns encoded (compressed) representation and a decoder that receives encoded data to reconstruct the original input. Recurrent Convolutional Autoencoder [51] and DZip [19] are few implementations of autoencoders [19, 34, 41, 51, 52]. These methods are not suitable for edge online compression since they are resource intensive [9] and require GPU, while some representatives also suffer from very low performance not sufficient for processing high frequency sensor data [19, 51].

Sequential methods combine several simple compression techniques sequentially. Huffman encoding, delta encoding, run-length encoding, and functional approximation techniques sequentially. These methods are generally adaptive and non-symmetric, either lossy or lossless.
encoding and Fibonacci binary encoding are mostly used techniques that serve as building blocks for more sophisticated methods. Delta encoding, run-length and Huffman [42], Sprintz [4], run-length binary encoding [43], RAKE [7], Gorilla [39] and Chimp [33] are examples of sequential methods. These techniques are lossless, symmetric, non-adaptive and computationally effective. However, lossy implementation of these techniques is also possible. The last category includes other time series compression methods that cannot be attributed to a single group. Major Extrema Extractor [14] or Continuous Hidden Markov Chain [23] belong to this category.

6.2 Lossless and Lossy Compression Techniques

There are some studies that perform a comparative analysis of lossless or lossy compression on time series. [22] propose a benchmark for error-bounded functional approximation methods by evaluating their compression factor, computation time, compression error and sensitivity to outliers. The following methods were evaluated: PCA (Piecewise Constant Approximation) [31], APCA (Adaptive PCA) [29], PWLH (Piecewise Linear Histogram) [5], SF (Slide Filter) [13], CHEB (Chebyshev Approximation) [3, 6] and GAMPS (Grouping and AMPlitude Scaling) [16]. The authors use 11 different real-life datasets that are very small in size recorded within the environmental project [46]. The largest dataset (i.e., humidity dataset) contains ~2M data points with 60s sampling interval (SI). They conclude that APCA and SF outperform other methods in terms of compression factor for most datasets. APCA and SF achieve at least 3x better computation time than the second-best methods SF and GAMPS. In terms of data quality, PCA and CHEB provide 2.78x lower RMSE than the other methods on average. GAMPS is found to be the least sensitive to outliers. The authors conclude that in general, APCA and SF show the best performance when a signal is not too noisy. They also note that the adaptability of a window size is an important factor to achieve high compression factor and low compression error. When the signal is very noisy, the authors recommend using GAMPS or APCA. In [49], the authors propose benchmarking lossless time series compression methods based on compression factor and time cost which includes compression and decompression time. TS_2DIFF [48], Gorilla [39], RLE [18], RLEB [44], RAKE [8] and Sprintz [4] are implemented in the distributed time series management system Apache IoTDB [47] for the evaluation. The methods are evaluated on 7 real-life datasets consisting of the application performance monitoring and web server logs. TS_2DIFF achieves superior performance for the datasets with large delta mean, while Gorilla performs better for the datasets with small delta mean and value variance than the other methods. In addition, Gorilla provides better encoding time and decoding time than TS_2DIFF. In conclusion, the authors suggest choosing the suitable compression method based on the properties of the dataset being compressed.

Although these studies provide comparative analysis of either lossy or lossless compression methods, they do not evaluate the methods in a scalable manner with massive real-life datasets and varying parameters. As an example, the most high-frequent dataset in [22], has 161004 data points with 2s SI, while we use a dataset with ~480M data points with 150ms SI.

7 CONCLUSION

In this paper, we show that MDB is a solution to the current challenges of managing high frequency sensor data from wind turbines across edge and cloud. We show that it efficiently addresses the challenges with: High Frequency Data, Limited Bandwidth, High Storage Costs, and Low Data Quality After Compression. MDB is compared to two solutions currently used in industry with three varying aspects.

We answer four RQs: RQ1: How does a high-frequency wind turbine dataset compress with the evaluated solutions? RQ2: How does MDB’s model types are used? RQ3: How is the transfer efficiency of the three solutions? RQ4: How well does MDB preserve the data quality of a high-frequency wind turbine dataset?

The results show that MDB outperforms LLC in terms of CF and transfer efficiency and matches AOG in terms of CF and transfer efficiency but adds orders of magnitude less error.

As an interesting insight we find that MDB’s multi-model compression is beneficial for compressing different datasets and the use of model types depends on the dataset, SI and ε. We also identified that the limited bandwidth from the edge to the cloud makes MDB’s ingestion time insignificant and that MDB’s compression provides a good compromise between efficient use of bandwidth and data quality. MDB’s model types lead to much smaller errors than allowed. The results in [10] show that MDB’s base model types with error bounds below 15%, produce compressed data that maintains the accuracy of forecasting methods. Thus, we conclude that MDB is an excellent solution for efficient management of wind turbine data across edge and cloud.

As future work, we will design new model types for MDB and implement them in the next version of ModelarDB [38] that is currently in development. In addition, we will improve the model fitting strategy by making it aware of properties of the time series.

ACKNOWLEDGEMENTS

This project is funded by the EU Horizon 2020 program under the Marie Skłodowska-Curie Innovative Training Networks (grant agreement No:955895) and the MORE project (grant agreement No:957345).

We thank Siemens Gases Renewable Energy for providing us access to real-life datasets and we thank Senior Data Scientist Emil Hedevang for insightful comments. We further thank our anonymous industry partner for providing access to real-life datasets.

REFERENCES


