Machine Learning Platform for Extreme Scale Computing on Compressed IoT Data

Seshu Tirupathi
IBM Research Europe
Dublin, Ireland
seshutir@ie.ibm.com

Dhaval Salwala
IBM Research Europe
Dublin, Ireland
dhaval.vinodbhai.salwala@ibm.com

Giulio Zizzo
IBM Research Europe
Dublin, Ireland
Giulio.Zizzo2@ibm.com

Ambrish Rawat
IBM Research Europe
Dublin, Ireland
Ambrish.Rawat@ie.ibm.com

Mark Purcell
IBM Research Europe
Dublin, Ireland
markpurcell@ie.ibm.com

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

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

Nguyen Ho
Aalborg University
Denmark
ntth@cs.aau.dk

Carlos E. Muniz Cuza
Aalborg University
Denmark
cemc@cs.aau.dk

Jonas Brusokas
Aalborg University
Denmark
jonasb@cs.aau.dk

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

Giorgos Alexiou
Athena Research Center
Greece
galexiou@athenarc.gr

Giorgos Giannopoulos
Athena Research Center
Greece
giann@athenarc.gr

Panagiotis Gidarakos
Athena Research Center
Greece
pgidar@athenarc.gr

Alexandros Kalimeris
Athena Research Center
Greece
akalimeris@athenarc.gr

Stavros Maroulis
Athena Research Center
Greece
stavmars@athenarc.gr

George Papastefanatos
Athena Research Center
Greece
gpapas@athenarc.gr

Ioannis Psarros
Athena Research Center
Greece
ipsarros@athenarc.gr

Vassilis Stamatopoulos
Athena Research Center
Greece
bstam@athenarc.gr

Manolis Terrovitis
Athena Research Center
Greece
mter@athenarc.gr

***Abstract*—With the lowering costs of sensors, high-volume and high-velocity data are increasingly being generated and analyzed, especially in IoT domains like energy and smart homes. Consequently, applications that require accurate short-term forecasts and predictions are also steadily increasing. In this paper, we provide an overview of a novel end-to-end platform that provides efficient ingestion, compression, transfer, query processing, and machine learning-based analytics for high-frequency and high-volume time series from IoT. The performance of the platform is evaluated using real-world dataset from RES installations. The results show the importance of high-frequency analytics and the surprisingly positive impact of error bounded lossy compression on machine learning in the form of AutoML. For example, when detecting yaw misalignments in wind turbines, an improvement of 9% in accuracy was observed for AutoML models on lossy compressed data compared to the current industry standard of 10-minute aggregated data. Thus, these small-scale experiments show the potential of the platform, and larger pilots are planned.**

***Index Terms*—Big Data, Cloud, Edge, Lossless Data Compression, Lossy Data Compression, Machine Learning, Renewable Energy Sources**

This work has been supported by the MORE project (grant agreement 957345), funded by the EU Horizon 2020 program.

I. Introduction

The widespread use of sensors and IoT devices in various domains like healthcare, energy, smart homes are generating huge volumes of time series data. Due to the high-volume and high-velocity of the data being produced, sensor data collected by monitoring Renewable Energy Sources (RES) installations are currently being stored as simple aggregates, e.g., 10-minute averages. This significantly reduces the amount of storage required at the possible cost of informative outliers and fluctuations. While analytics on aggregated time series work for some use cases, there are many applications like time-critical failure detection, very short-term forecasting, real-time anomaly detection, etc where analytics on the raw high-frequency time series is potentially required, e.g., [1], [2].

In addition, as the frequency at which the raw sensor data is being generated increases, so does the challenge of collecting and managing the raw high-frequency time series. Thus, limiting the tools that can be provided for forecasting, prediction, and diagnostics [3]. The lack of suitable tools and techniques can be explained by the following four main factors: (a) Volume – the size of the data is already bigger than almost any other domain, (b) Velocity – the rate at which

data is generated is extremely high, since sensors sample and report information at very high frequencies, (c) Variety – the data exhibit a high degree of complexity when sequence, timing order, and location are taken into consideration and the features of interest are broad and diverse in different contexts, (d) Veracity – reliability of sensor data is often questionable, e.g., due to noise, communication failures, and faulty sensors.

The key objective of the MORE platform (Management of Real-time Energy data) is to create a platform for time series management, query processing, and machine learning (ML)-based analytics that will expand the current state-of-the-art for time series management systems (TSMSs) [4], [5] and provide a complete platform for efficiently managing time series from RES installations. In this paper, we provide an overview of the MORE platform we currently are developing to efficiently manage and analyze high-frequency time series from RES installations. We also provide an overview of the platform's core algorithms and components, e.g., model-based lossless compression, model-based lossy compression, and ML algorithms for problems that are constrained by the complexities mentioned above. As the current paper provides a high-level overview of the MORE platform, references to publications with more detailed information are also provided. Finally, while the platform is currently being built with management and analytics of high-frequency time series from RES installations as its core use case, it is general enough to be used in other domains that require efficient management and analytics of high-frequency time series. In summary, through the development of the MORE platform, we have made the following contributions:

- **(I)** Performed an extensive survey of TSMSs developed through academic or industrial research [5].
- **(II)** Analyzed the requirements and limitations for data management and analytics of high-frequency time series for RES installations on the edge and extended ModelarDB to efficiently manage high-frequency time series across the edge and the cloud [6].
- **(III)** Developed machine learning and pattern extraction libraries and API framework for handling compressed data.

The structure of the paper is as follows. Section II provides a short literature survey of TSMSs the supports lossy compression and the effect of lossy compression on ML algorithms. The architecture of the MORE platform is provided in Section III. Then, an evaluation of the MORE platform is provided in Section IV. Finally, our conclusion and future work are provided in Section V.

II. Related work

There are two main aspects to consider when developing extreme scale computing platforms for IoT: a) Efficient storage of the time series through lossy compression as it provides significantly lower storage requirements than lossless compression [7] and subsequently efficient querying of the lossy compressed time series. b) The performance impact of using lossy compressed time series for ML algorithms.

Querying lossy compressed time series: A significant number of TSMSs that support executing queries directly on compressed representations of time series have been proposed [4], [5]. TimeTravel [8] extends PostgreSQL with support for fitting models to time series and executing queries on these models. The use of models allows the system to provide a uniform interface for exact and approximate queries on historical data as well as forecasting. SummaryStore [9] stores time series using multiple different representations that all can be combined as the data ages. This reduces the amount of storage required but adds additional error. During query processing, the query result may be approximate depending on the representations used. M-DB [10] supports executing queries on missing or out-of-order data points through user-defined prediction methods. Tristan [11], [12] compresses time series using dictionary encoding and supports executing some queries directly on the compressed representation. Tristan's compression method was extended to take time series correlation into account [13]. tspDB [14] extends PostgreSQL with support for time series imputation and forecasting using an incremental prediction method based on matrix factorization. ModelarDB [15] fits multiple different types of models to each time series as its structure changes over time. Each dynamically sized sub-sequence is stored using the type of model that provides the best compression ratio. ModelarDB supports executing queries on the models and reconstructed data points using two different views. ModelarDB was extended to compress groups of time series with similar values together [16], [17] and to transfer data from edge to cloud [6].

Impact of lossy compression on ML algorithms: As lossy compression is becoming more broadly used in TSMSs [4], [5], it becomes necessary to develop a profound understanding of the impact lossy compression has on ML algorithms. The impact of lossy compression on change detection was analyzed in [18]. Specifically, it was investigated how multiple combinations of compression and change detection algorithms perform on different datasets showing that precise detection is possible even on heavily compressed data. The impact of the lossy compression error on time series forecasting was analyzed in [19] and actually showed a slight improvement in the results for small error bounds. A similar study for time series classification showed an improvement in the accuracy of classification when compressed data is used [20]. Although further analysis needs to be conducted [21], these results suggest that lossy compression can be used to efficiently manage high-volume and high-velocity time series data produced by RES installations while also maintaining the accuracy of the ML algorithms that consume the data and in some cases even improve it.

To summarize, while there are isolated solutions for each of the extreme-scale computing challenges that prevent management and analytics of IoT data at scale, there is a clear need for a holistic solution that seamlessly integrates management of time series through lossy compression and ML algorithms on data compressed using lossy compression in a unified platform.

III. Architecture of the MORE Platform

The architecture of the MORE platform is shown in Fig. 1 and consists of five open-source libraries and systems that are being developed/enhanced for the platform:

- ModelarDB[1] – A TSMS designed to efficiently manage high-frequency time series through lossy compression. It provides management of data on the edge, transfer of data to the cloud, and management of data in the cloud.
- SAIL (Streams and Incremental Learning)[2] – Incremental learning library for big data. Also supports neural network batch learning models.
- MoreUtils[3] – API library for interacting with ModelarDB and ML life-cycle components.
- more-pattern-extraction[4] – Library for identifying interesting patterns (motifs, change-points, deviations) on RES time series.
- complex-event-detection[5] – Library for detecting behaviors and deviations that signify potential issues on RES time series.

These five libraries and systems cover all the aspects of the IoT data lifecycle such as ingesting the sensor data on the edge nodes, lossy compression, transfer to the cloud, and analytics using ML algorithms on lossy compressed data in the cloud. These steps are explained in detail below.

To manage the large amount of high-frequency time series, the platform uses and extends the TSMS ModelarDB [6], [15]–[17]. The TSMS ingests and compresses the time series on the edge using *model-based lossy compression*. In the context of ModelarDB, a model is any representation that can reconstruct the values for a dynamically-sized sub-sequence of the original time series within a user-defined per-value error bound (possibly 0%). For example, the linear function $f(t) = 0.03t + 2.5$ is a model as shown in Fig. 2. Instead of storing the raw time series, ModelarDB only stores the coefficients for the models and enough metadata to reconstruct the raw time series within the user-defined error bound (possibly 0%). The user-defined error bound and the length of the sub-sequences are generally positively correlated, thus additional compression can be achieved if the error bound is increased. However, in contrast to using simple aggregates for compression, since ModelarDB uses a per-value error bound, important fluctuations and outliers are preserved if they exceed the error bound. As stated, ModelarDB also supports 0% error bounds such that lossless compression is possible. A maximum length for the sub-sequences can also be set for some models.

Concretely, raw sensor readings are first received from an MQTT broker. A model is then *fitted* to the received data points using a set of model types. ModelarDB currently includes three different model types: constant [22], linear [23], and lossless [24]. A set of model types is used as the structure of time series very often changes over time. ModelarDB generally provides better compression than existing storage formats when using the included model types [15], [17]. However, it is very simple for users to optionally add additional model types which are optimized for their domain by implementing two simple Java interfaces (`ModelType` and `Segment`) and pointing to the resulting classes in ModelarDB's configuration. For additional details see [6]. When a new data point is received that none of the model types can fit a model to, i.e., when the error bound or length bound would be exceeded, the model that provides the best compression ratio for the previous data points is emitted and new models are fitted to the forthcoming data points. As soon as a model is emitted, it can be used to answer queries. Thus, ModelarDB's ingestion method provides both very a high degree of compression within a user-defined error bound and support for real-time ingestion of high-frequency sensor data using the limited hardware resources that are available on the edge nodes. On the edge nodes, the metadata and models are stored in the RDBMS H2, as columnar Apache ORC files, or as columnar Apache Parquet files. For additional details about ModelarDB's file-based data store see [6].

When the amount of compressed data, i.e., metadata and models, reaches a user-defined size, it is transferred by ModelarDB to the cloud. Since bandwidth can be limited for RES installations in remote locations, e.g, 500 Kbits/s to 5 Mbits/s, a very high level of compression is needed to transfer the high-frequency time series. ModelarDB's data transfer component uses Apache Arrow Flight in order to efficiently transfer the compressed data. On the cloud, data is stored in Apache Cassandra, columnar Apache ORC files, or columnar Apache Parquet files. The current version of the data transfer component simply transfers the metadata and models in batches. However, multiple optimizations are planned. (i) The data transfer component should optimize use of bandwidth such that urgent data is transferred immediately, while less critical data are transferred in bulk later when bandwidth is available. (ii) Shared prediction should be applied on the edge and in the cloud to reduce the amount of bandwidth used and improve latency. (iii) When similar models are received in the cloud, they should be combined if possible within the error bound to further reduce the amount of storage required.

In addition to its model-based compression, ModelarDB has also been extended to efficiently support *derived time series*, i.e., time series that can be computed from other time series. For example, when angles are measured over time, practitioners often store both a time series with the raw measurements in the form (t, v) and a similar time series with data points of the form $(t, \sin(v \times \pi/180))$. Instead of physically storing the latter, ModelarDB can store information about the transformation to apply, similar to a view in an RDBMS. To ensure the derived time series are efficiently computed, dynamic code generation is used to generate Java bytecode for the functions at runtime. For additional details about derived time series see [6].

ModelarDB uses SQL as its query language and exposes

[1] https://github.com/skejserjensen/ModelarDB
[2] https://github.com/IBM/sail/
[3] https://github.com/IBM/more-utils/
[4] https://github.com/MORE-EU/more-pattern-extraction
[5] https://github.com/MORE-EU/complex-event-detection

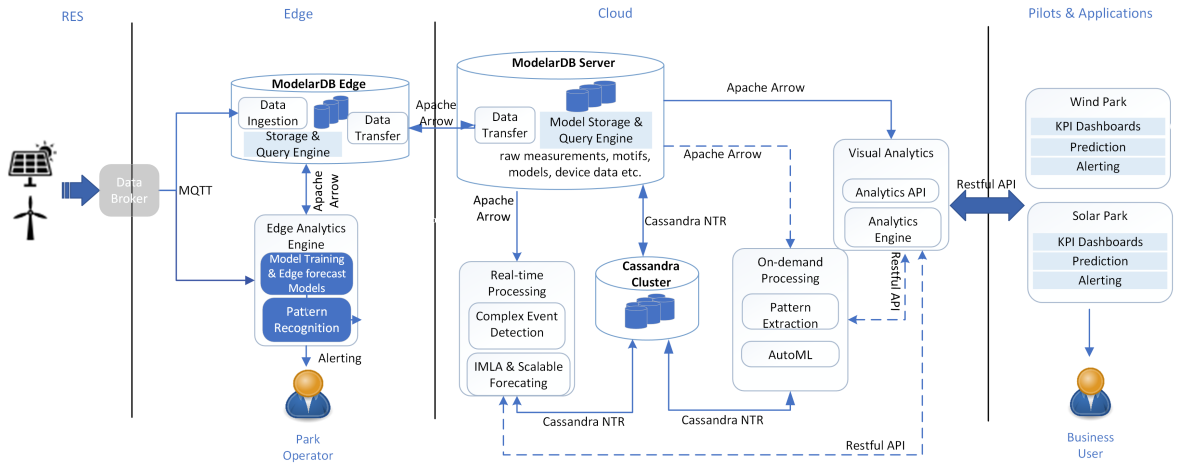

Fig. 1. Architecture diagram of MORE platform.

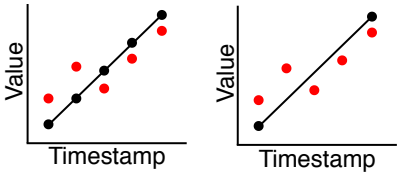

Fig. 2. Computation of SUM for a model of type Swing using the Data Point View (Left) and the Segment View (Right). The data points ModelarDB originally ingested are Red, while the reconstructed data points are Black.

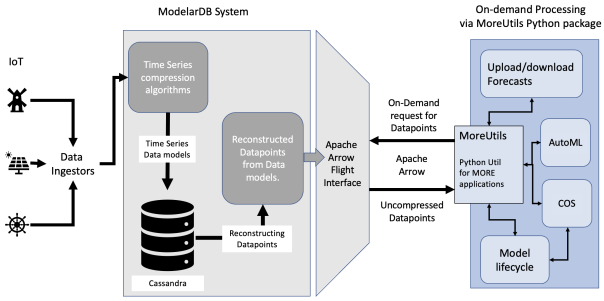

Fig. 3. MoreUtils component architecture.

the stored time series at two different levels through the *Data Point View* and the *Segment View*. The Data Point View exposes the time series as data points reconstructed from the metadata and models so arbitrary SQL queries can be executed. While the Data Point View can support arbitrary SQL queries, many aggregates can be computed much more efficiently directly from the metadata and models. To do so, ModelarDB exposes the time series as metadata and models through the Segment View and provides a selection of User-Defined Functions (UDFs) and User-Defined Aggregate Functions (UDAFs) for computing aggregates directly from the metadata and models. An example that computes SUM for a linear model of type Swing is shown Fig. 2. For the Data Point View (left), SUM is computed in linear time by summing the values reconstructed from the metadata and model using the equation $f(t) = a \times t + b$ where the result and t are the value and timestamp of each data point, respectively. For the Segment View SUM (right), SUM is computed in constant time for the model by computing the value of the first data point represented by the metadata and model v_1, computing the value of the last data point represented by the metadata and model v_n, and then computing SUM as $((v_1 + v_n)/2) \times n$ where n is the number of data points represented by the model.

After the compressed data has been transferred to the ModelarDB instance deployed in the cloud, clients can query it through an HTTP or Socket interface that returns JSON or an Apache Arrow Flight interface that returns Apache Arrow. To simplify retrieving data for analytics and visualization MoreUtils can be used as shown in Fig. 3. The Blue Box represents the client infrastructure of which MoreUtils is a part. MoreUtils is an API library written in Python to support MORE applications with operations related to the high-frequency time series data, such as retrieving compressed data points from ModelarDB and maintaining the ML model's lifecycle. It uses the PyArrow interface by default via PyModelarDB to efficiently retrieve data points from ModelarDB. However, the HTTP and Socket interfaces are also supported. Using these interfaces, MoreUtils makes it seamless to retrieve decompressed data points from ModelarDB. When a client requests data points via MoreUtils, a matching request is sent to ModelarDB. ModelarDB then rewrites the predicates in the query so they can be pushed to the data store it is configured to use, in this example Apache Cassandra, retrieves at least the required metadata and models from Apache Cassandra, reconstructs at least the requested data points, and then transfers the requested data points to the client. When it receives the data points, MoreUtils performs several operations, such as joining the univariate time series by their timestamps to create a multivariate time series and labeling the resulting value columns based on their time series id. MoreUtils also

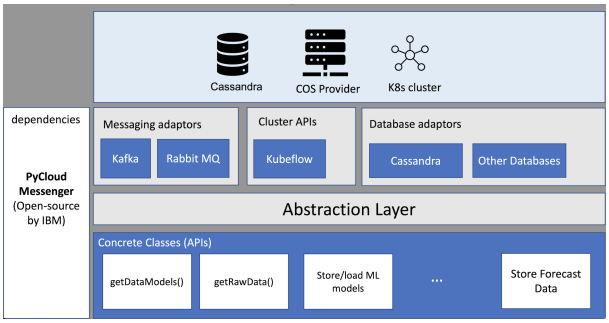

Fig. 4. Machine learning model APIs and model life-cycle management.

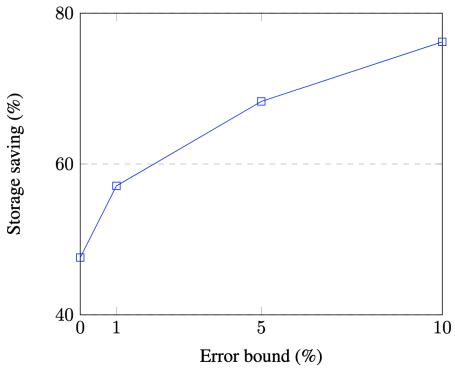

Fig. 5. % reduction in the amount of storage used when using ModelarDB.

provides APIs for reading time series one data point at a time or as batches of data points. Several data filters are available to suit specific use cases. Additionally, MoreUtils provides coherent APIs to manage the ML model lifecycle as shown in Fig. 4. For example, it provides functionality for uploading and downloading ML models to/from Cloud Object Service (COS). The forecasts made by ML models can also be stored in one of the several data stores like Apache Cassandra.

The Pattern Extraction and Complex Event Detection components also exploit the aforementioned APIs to use ML and data mining techniques in order to detect patterns of particular significance for the RES use case. Specifically, they allow the offline extraction of interesting patterns that may denote a specific phenomenon, e.g. soiling on solar panels, or a changepoint, e.g. a solar panel washing event, as well as the online detection of patterns and change-points, e.g. that Yaw Misalignment (YM) has taken place on a wind turbine.

Similarly, batch and incremental models are supported through a common set of APIs. Specifically, Scikit-learn based *fit*, *partial_fit*, and *predict* models within the SAIL library for regression, classification, anomaly detection, and clustering for time series data. Standard Scikit-learn based AutoML algorithms can also be run for the models available in SAIL. As stated, ML and pattern extraction models retrieve reconstructed data points from ModelarDB using the MoreUtils package.

IV. Evaluation

The MORE platform is evaluated using real-world datasets from RES installation to study the effect ModelarDB's lossy compression and the downstream impact of the compressed data on ML algorithms. First, we used ModelarDB to ingest a real-world RES dataset to evaluate the effect of its lossy compression and support for derived time series on the storage cost. The original dataset consists of 180 time series stored as Apache ORC files (6.3 GiB) of which 72 are derivable. As shown in Fig. 5, even with a 0%-error bound, the storage cost is reduced by 47.62%. Similarly, the amount of storage required reduces by 57.14%, 68.25%, and 76.19% for a 1%-, 5%-, and 10%-error bound, respectively. This significant reduction makes it possible to store and analyze much larger datasets than other RDBMs, and TSMSs. Also, applying the compression already on the edge makes it possible to transfer much larger datasets over connections with very limited bandwidth. For more detailed evaluations of ModelarDB's ingestion speed, compression, and query performance see [15] and [17]. For example, it was shown in [17] that ModelarDB provides up to 13.7x higher ingestion speed, requires up to 113x less storage, and can execute aggregate queries up to 573x faster than InfluxDB, Apache Cassandra, Apache Parquet, and Apache ORC.

Next, we used real-life data from wind turbines to evaluate the effect on AutoML algorithms when the data is compressed by ModelarDB at different error bounds for yaw misalignment. YM has been identified as an important factor that reduces the efficiency of wind parks. Wind turbines are aligned with inflow wind direction to maximize energy generation. However, YM is a phenomenon in wind turbines that results in significant power reduction. ML models have been developed in this section to detect YM in one hour windows where high-frequency data can provide better insights [25].

Specifically, we considered high-frequency (HF) data (4 sec), 10 minute aggregates (10 min agg), 1 minute aggregates (1 min agg), and data compressed by ModelarDB with the error bounds {2%, 5%, 10%, 20%}, respectively. The dataset contains data from June 6th, 2018, to March 12th, 2019, at 4 second frequency. The dataset has the following six attributes as features: wind speed, pitch angle, rotor speed, active power, nacelle direction, and wind direction. The data set has active power as the target variable. The model is trained to predict active power based on the features provided in the dataset between February 12th and March 12th where no YM was detected. The data was transformed using techniques described in [26].

The difference between 1 hour aggregated predicted active power and observed active power was taken and normalized with the average active power observed for the training period. Since YM results in reduced active power, a user-defined parameter defines the threshold for the difference in aggregated active power to detect YM. The parameter is set to 0.08 in the experiments defined below. Thus, the YM problem can be formalized as a trivial binary classification task using this threshold parameter. Multi-layer Perceptron (MLP) models

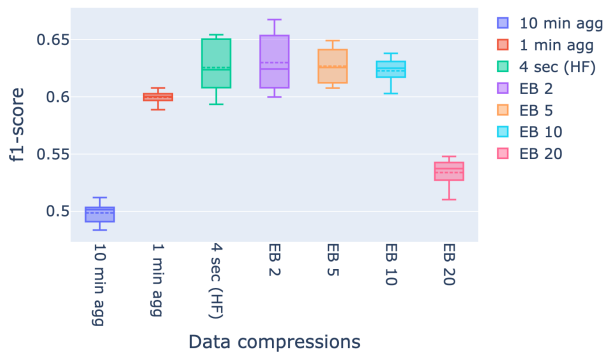

Fig. 6. f1-score of the best AutoML model to detect YM when using different compression methods and parameters. EB is a shorthand for error bound.

were trained to predict active power with the following parameters: *hidden_layer_sizes* – [[40,40], [50,50], [100,100], [20,20]] and *activation* – ['relu' ,'tanh', 'logistic'].

Finally, AutoML (based on GridSearchCV on the transformed dataset) was used to find the best model for each combination of compression method and parameter. The experiment was repeated 10 times and the average results are presented. The R2 score and RMSE for all combinations of compression methods and parameters are shown in Table I and show that the best model from AutoML is able to learn to a high accuracy across all combinations of compression methods and parameters. However, the accuracy of the models differs across the different combinations of compression methods and parameters as shown in Table II. The best combination of compression methods and error bound have an accuracy of 9 percentage points higher than the 10 minute aggregated data which is the current industry standard [27]. This underlines the importance of high-frequency data for IoT applications and that data compressed using lossy compression can be used to detect YM.

Fig. 6 shows the f1-score for the best models trained using data compressed using different combinations of compression methods and parameters. As can be observed, the accuracy of the model trained using data compressed by ModelarDB with an error bound equal to or less than 10% are much more accurate than the model trained using 10 minute averages. The models also become more stable as the error bound increases from raw data to an error bound of 10% with the interquartile range getting narrower. This is probably because the noise in the data gets smoothed to a higher degree when the error bound is increased. Finally, it is important to note that the accuracy of the model trained on data compressed with a 10% error bound is better than the model trained using 1 min aggregates as well. The 1 min aggregates on the other hand require slightly less storage than when the data set is compressed with a 10% error bound, approximately the same amount of storage as when ModelarDB uses a 14-16% error bound. However, as more data is ingested from the wind park, using simple 1 minutes aggregates for compression means the amount of storage used should increase close to linearly while the amount of storage used by ModelarDB should decrease at an increasingly lower rate due to the planned functionality for *combining similar models from different wind turbines*. Model types specifically optimized for this wind turbine data set could also be added to reduce the amount of storage required. Finally, ModelarDB's per-value error bound guarantees that important outliers and fluctuations that exceed the error bound are preserved, while the 1 min aggregates do not.

TABLE I
R2 SCORE AND RMSE FOR THE TEST SET USING THE BEST AUTOML MODEL AVERAGED ACROSS 10 RUNS FOR VARIOUS TYPES OF COMPRESSIONS.

data	R2	RMSE
10 min agg	0.96	61.79
1 min agg	0.97	52.43
4 sec (HF)	0.93	70.63
EB 2%	0.93	73.62
EB 5%	0.94	62.10
EB 10%	0.93	96.83
EB 20%	0.95	70.97

TABLE II
AVERAGE MODEL METRICS OF THE BEST AUTOML MODELS USED TO DETECT YM ACROSS 10 RUNS FOR VARIOUS TYPES OF COMPRESSION.

data	precision	f1-score	accuracy
10 min agg	0.85	0.50	0.58
1 min agg	0.88	0.60	0.65
4 sec (HF)	0.85	0.63	0.67
EB 2%	0.82	0.63	0.67
EB 5%	0.76	0.63	0.65
EB 10%	0.77	0.63	0.65
EB 20%	0.78	0.53	0.60

V. CONCLUSION AND FUTURE WORK

We presented an overview of a novel extreme scale computing platform for compressed IoT data. This work shows the potential of scalability of the platform as more and more sensors are added where *model-based compression* of time series and *derived time series* can significantly reduce the amount of storage required while still maintaining the required accuracy for advanced ML-based analytics. incremental machine learning algorithms have also been implemented considering the size of the data.

For future work, AutoML on streaming data remains an open challenge and will be explored. Also, the system components and APIs will be developed such that the ML models, Pattern Extraction, and Complex Event Detection components can use the models created by ModelarDB directly as features instead of reconstructed data points.

REFERENCES

[1] Circonus, "Smart grid," https://www.circonus.com/solutions/smart-grid/, 2022, accessed: 2022-08-11.

[2] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, "NILMTK: An open source toolkit for non-intrusive load monitoring," in *Proceedings of the 5th International Conference on Future Energy Systems*, ser. E-Energy '14. ACM, Jun. 2014, pp. 265–276.

[3] S. Khare and M. Totaro, "Big data in iot," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2019, pp. 1–7.

[4] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time Series Management Systems: A Survey," *IEEE Trans. Knowl. and Data Eng.*, vol. 29, no. 11, 2017.

[5] ——, "Time Series Management Systems: A 2022 Survey," in *Data Series Management and Analytics (Forthcoming)*, T. Palpanas and K. Zoumpatianos, Eds. ACM.

[6] S. K. Jensen, C. Thomsen, and T. B. Pedersen, "ModelarDB: Integrated Model-Based Management of Time Series from Edge to Cloud," *Trans. Large-Scale Data- and Knowledge-Centered Syst.*, vol. (Forthcoming).

[7] N. Q. V. Hung, H. Jeung, and K. Aberer, "An Evaluation of Model-Based Approaches to Sensor Data Compression," *IEEE Trans. Knowl. and Data Eng.*, vol. 25, no. 11, pp. 2434–2447, 2013.

[8] M. E. Khalefa, U. Fischer, T. B. Pedersen, and W. Lehner, "Model-based Integration of Past & Future in TimeTravel," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 1974–1977, 2012.

[9] N. Agrawal and A. Vulimiri, "Low-Latency Analytics on Colossal Data Streams with SummaryStore," in *Proc. 26th ACM Symp. on Operating System Principles*. ACM, 2017, pp. 647–664.

[10] V. Arora, M. J. Amiri, D. Agrawal, and A. E. Abbadi, "M-DB: A Continuous Data Processing and Monitoring Framework for IoT Applications," in *2019 Int. Conf. on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*. IEEE, 2019, pp. 1096–1105.

[11] A. Marascu, P. Pompey, E. Bouillet, O. Verscheure, M. Wurst, M. Grund, and P. Cudre-Mauroux, "MiSTRAL: An Architecture for Low-Latency Analytics on Massive Time Series," in *Proc. 2013 IEEE Int. Conf. on Big Data*. IEEE, 2013, pp. 15–21.

[12] A. Marascu, P. Pompey, E. Bouillet, M. Wurst, O. Verscheure, M. Grund, and P. Cudre-Mauroux, "TRISTAN: Real-Time Analytics on Massive Time Series Using Sparse Dictionary Compression," in *Proc. 2014 IEEE Int. Conf. on Big Data*. IEEE, 2014, pp. 291–300.

[13] A. Khelifati, M. Khayati, and P. Cudré-Mauroux, "CORAD: Correlation-Aware Compression of Massive Time Series using Sparse Dictionary Coding," in *Proc. 2019 IEEE Int. Conf. on Big Data*. IEEE, 2019.

[14] A. Agarwal, A. Alomar, and D. Shah, "tspDB: Time Series Predict DB," in *NeurIPS 2020 Competition and Demonstration Track*. PMLR, 2020, pp. 27–56.

[15] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "ModelarDB: Modular Model-Based Time Series Management with Spark and Cassandra," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1688–1701, 2018.

[16] ——, "Demonstration of ModelarDB: Model-Based Management of Dimensional Time Series," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*. ACM, 2019, pp. 1933–1936.

[17] ——, "Scalable Model-Based Management of Correlated Dimensional Time Series in ModelarDB$_+$," in *Proc. 37th Int. Conf. on Data Engineering*, 2021.

[18] G. Hollmig, M. Horne, S. Leimkühler, F. Schöll, C. Strunk, A. Englhardt, P. Efros, E. Buchmann, and K. Böhm, "An evaluation of combinations of lossy compression and change-detection approaches for time-series data," *Information Systems*, vol. 65, pp. 65–77, 2017.

[19] F. Eichinger, P. Efros, S. Karnouskos, and K. Böhm, "A time-series compression technique and its application to the smart grid," *The VLDB Journal*, vol. 24, no. 2, pp. 193–218, 2015.

[20] A. Moon, J. Kim, J. Zhang, and S. W. Son, "Evaluating fidelity of lossy compression on spatiotemporal data from an iot enabled smart farm," *Computers and Electronics in Agriculture*, vol. 154, pp. 304–313, 2018.

[21] F. Cappello, S. Di, and A. M. Gok, "Fulfilling the promises of lossy compression for scientific applications," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2020.

[22] I. Lazaridis and S. Mehrotra, "Capturing Sensor-Generated Time Series with Quality Guarantees," in *Proceedings of ICDE*. IEEE, 2003.

[23] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref, and W. Zwaenepoel, "Online piece-wise linear approximation of numerical streams with precision guarantees," *PVLDB*, vol. 2, no. 1, 2009.

[24] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan, "Gorilla: A Fast, Scalable, In-Memory Time Series Database," *PVLDB*, vol. 8, no. 12, pp. 1816–1827, 2015.

[25] Thomas van Delft, "Yaw optimization using high-frequency wind turbine data," https://dnv.com/article/yaw-optimization-using-high-frequency-wind-turbine-data-186379, 2022, accessed: 2022-08-11.

[26] L. Gao and J. Hong, "Data-driven yaw misalignment correction for utility-scale wind turbines," *Journal of Renewable and Sustainable Energy*, vol. 13, no. 6, p. 063302, 2021.

[27] E. Gonzalez, B. Stephen, D. Infield, and J. Melero, "On the use of high-frequency scada data for improved wind turbine performance monitoring," in *Journal of Physics: Conference Series*, vol. 926, no. 1. IOP Publishing, 2017.