Modeling Large Time Series for Efficient Approximate Query Processing

Kasun S. Perera[1](✉), Martin Hahmann[1], Wolfgang Lehner[1], Torben Bach Pedersen[2], and Christian Thomsen[2]

[1] Database Technology Group, Technische Universität Dresden, Dresden, Germany
{kasun.perera,martin.hahmann,wolfgang.lehner}@tu-dresden.de
[2] Department of Computer Science, Aalborg University, Aalborg, Denmark
{tbp,chr}@cs.aau.dk

Abstract. Evolving customer requirements and increasing competition force business organizations to store increasing amounts of data and query them for information at any given time. Due to the current growth of data volumes, timely extraction of relevant information becomes more and more difficult with traditional methods. In addition, contemporary Decision Support Systems (DSS) favor faster approximations over slower exact results. Generally speaking, processes that require exchange of data become inefficient when connection bandwidth does not increase as fast as the volume of data. In order to tackle these issues, compression techniques have been introduced in many areas of data processing. In this paper, we outline a new system that does not query complete datasets but instead utilizes models to extract the requested information. For time series data we use Fourier and Cosine transformations and piecewise aggregation to derive the models. These models are initially created from the original data and are kept in the database along with it. Subsequent queries are answered using the stored models rather than scanning and processing the original datasets. In order to support model query processing, we maintain query statistics derived from experiments and when running the system. Our approach can also reduce communication load by exchanging models instead of data. To allow seamless integration of model-based querying into traditional data warehouses, we introduce a SQL compatible query terminology. Our experiments show that querying models is up to 80 % faster than querying over the raw data while retaining a high accuracy.

1 Introduction

In the current age of information, data is generated at an unprecedented state and stored in increasing volume. As today's economy and society run on knowledge, organizations not only need to keep massive amounts of data but also must be able to extract information from them. This concerns every imaginable area of modern life from stock market fluctuations and news feeds in social networks to renewable energy supply and vehicle travel patterns in traffic control systems. As the volume of gathered data grows bigger in all these application scenarios

DOI: 10.1007/978-3-319-22324-7_16

two major problems emerge. First, data must be kept in order to utilize it, which requires sophisticated storage systems. Second, querying the data becomes more demanding and poses new challenges for the information extraction process.

While the type of generated data varies with each application domain, time series are one of the most common formats of data as they capture the change/behaviour of certain measures/objects over time. This is why analyzing and querying time series data is an important part in many decision making processes. For example, a company that sells certain products is interested in sales patterns occurring in the past in order to make decisions about the future. Such queries often include average values, counts and sums for items in certain periods of time or geographic regions. In order to obtain the most up-to-date information, these queries are issued repeatedly which means large amounts of data must be loaded, copied and processed frequently. As data volume increases while computational resources are often constrained, query processing of large data becomes inefficient with regards to time. Although the answers that are generated are exact, decision making often favours an approximate answer at the right time over an exact one that is available too late. Before introducing our model-based database system we would like to define what a model is in our context. *A model is a representation, generally a simplified description, especially a mathematical one, of a system or process to assist in calculations and predictions* [5]. As an example systematic sample from a Fast Fourier Transformation (FFT) of a complex signal can be seen as a model of the original signal.

Contemporary literature suggests different methods for different approximate queries and generally associate one model with one or more queries. This approach limits the utilization of models as a model designed for one query cannot be used for a different query. In this work we propose a system that maintains a pool of models within database itself such that queries can either be answered with individual models or with a combination of 'general-purpose' models. By utilizing a model-based representation of the underlying data we are able to generate approximate answers much faster than exact answers from the original data. In addition our representative models have a small memory footprint and still provide query results with high accuracy. In the remainder of this paper we first outline the concept of our system, then describe early results for model creation and querying on time series data and finally give an outlook regarding the future potential of our approach.

2 Model-Based Database System Concept

It is obvious that large volumes of data can significantly reduce the runtime efficiency of complex queries in traditional database systems. One approach to tackle this issue is to materialize query results. While being a useful approach, query materialization does not provide full flexibility and scalability as materialized queries have to be updated whenever there are changes to the query or the data. To overcome this limitation while still increasing query performance we propose a model-based database system which runs queries against representative models to produce approximate results. This offers multiple benefits:

(1) models have a smaller memory footprint than the raw data, (2) maintaining a pool of different models for the same data allows to select an optimal model for each issued query, and (3) models can be used to re-generate the approximated original data with high accuracy. In this section we outline the concept of a system that enhances query efficiency by representing data with models and executing queries over these models.

When a query is issued against our model-based database system, the database engine decides whether to use a model or the conventional database approach to answer the query. The conventional approach is to scan and fetch the required raw data from the storage (usually disk storage) and execute the query over the dataset. If the decision is to use models, an appropriate stored model is accessed and approximate results are produced. Figure 1 depicts the overall architecture of the query processing system. After the query engine decides that a query will be answered with a model, it selects those models that produce the best possible results for the given query. In order to make this decision, the system maintains statistics of past queries that were executed on both models as well as raw data. This means, our proposed system undergoes an initial learning phase before it can fully apply the model based approach. During this phase, query statistics for queries running over raw data as well as models are recorded.

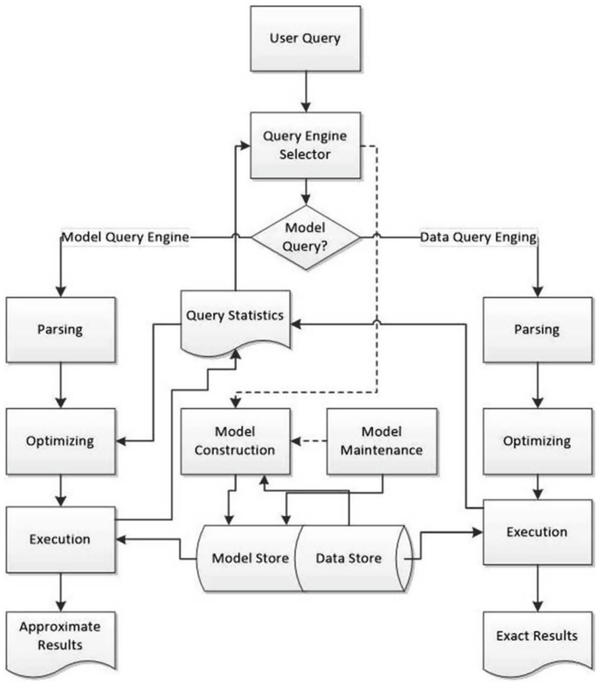


Fig. 1. Overall system architecture with both model and data query engines

The major part of the result-generation process takes place in the query processor. It is responsible for parsing the user query, extracting query components, optimizing the operations and finally executing the optimized query. Similar to the data query processor, the first step of the model query processor is to parse the user query into identifiable query objects. During parsing it is necessary to identify parts that relate to model query processing (optimization and execution) and other query operations. The proposed query parser extracts query objects such as Projections (*SELECT avg(productsales)*), Source Relations (*FROM sales*), Selections (*WHERE product='xxx'* or *WHERE date BETWEEN start to end*), Model Selections (*USE MODEL modelCategory*) as well as error- and time-bound parameters. An example of such a query is given below:

```
SELECT avg(sales)
FROM sales
WHERE date BETWEEN 01-01-2010 to 31-12-2013
USE MODEL modelCategory
ERROR WITHIN 10 %
RUNTIME WITHIN 5 SECONDS;
```

One of our main goals is the seamless integration of models into the database system in order to enable users to perform the same SQL query that they would run on raw data. In addition we add specific parameters to the query syntax namely TIMEOUT, ACCURACY, and MODEL. A sample query syntax that employs these new parameters is shown above. Our taxonomy is similar to [1] and [6]. The given example shows that users can explicitly define which model should be used to answer the query. In addition the user can define an error bound for the results. This directly affects query performance as approximate results can be generated from high-level models with less accuracy in a very short time, or from low-level models with high accuracy at the cost of a longer running time. In our model hierarchy, high-level models refer to coarse grained models with low accuracy and low-level model with high accuracy that represent the original data with fine granularity.

An important part of our proposed system is the query optimization which uses stored query statistics to decide the optimal set of models to answer a query within given time and accuracy constraints.

3 Model Querying for Time Series Data

While we focus on the modeling of time series data in this paper, our system can be extended to other types of data with different sets of models. In this section we describe an early stage of our system for time series data, where we focus on query efficiency and accuracy on large time series.

Our system employs a model pool where different models, built on the original data, are stored to answer user queries. Our goal is to store multiple models of the same data, so that queries with different parameters can be answered by a stored model or combination of them. For simplicity we define our time series as $TS = (t_1, v_1), (t_2, v_2), ..., (t_n, v_n)$ where TS defines the whole time series and (t_x, v_x) defines an individual (time, value) pair of the time series. A final model built on the original time series TS is referred to as $\Sigma(TS)$ and is a collection of sub-models of $\sigma(ts)$ such that each represents a smaller part of the larger time series. The transformation of TS into the model $\Sigma(TS)$ is described in the following sub section.

3.1 Model Construction

In the current version of our system, we use the described approaches to represent and query large time series data. In this section we describe how we construct a model pool to represent time series data. While we focus on individual time series in this paper, we plan to extend our model pool for multiple time series so that one model can represent different similar time series. We focus on large time series with millions of points. Generally, different patterns emerge in different periods of long time series. This is why it is necessary to consider a large time series as a collection of continuous smaller time series, which in general show more manageable patterns within a given range. Models derived from these sub-time-series have a higher accuracy as they only need to represent a compact set of local changes. The model construction process that turns the original time series into a model is depicted in Fig. 2. It starts by decomposing the long original time series into smaller parts and then models each part using the modeling approaches we describe in the next section. Each part of the original time series is considered as a single model and stored in a matrix along with a partition ID that indicates the part of the original time series the model represents.

The models created that way are then combined to generate query results. When a given time series TS is partitioned into smaller parts $(ts_1, ts_2, ...)$, each part is going through a model construction phase. In this phase, an initial model is constructed on each partition ts_i and passed on to the model evaluation. There, it is tested for accuracy against the original data and added to the model pool if it reaches a user-defined accuracy level. If the model fails the accuracy test, an update is performed in order to adjust model parameters and increase accuracy. If a model fails to maintain adequate accuracy bounds then raw data for that

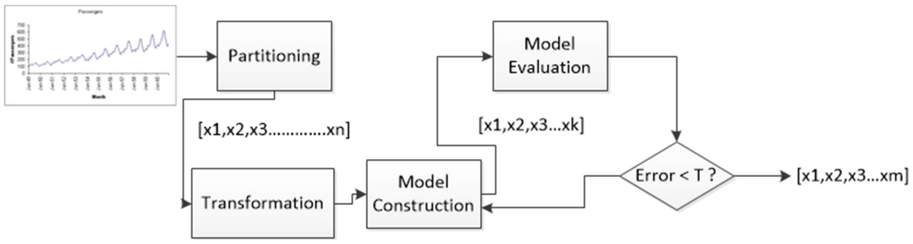


Fig. 2. Step-by-step construction of models from time series.

particular part of the time series is stored. The sequence of model evaluation and update is repeated until the given accuracy criteria are fulfilled. In the following, we describe the set of representative models that we use to transform TS into $\Sigma(TS)$.

Aggregation over Time Granularity. Segmenting a time series into several meaningful smaller parts and using these individual parts for queries over the original time series is a widely used approach in many applications. An example of a segmentation-based system for similarity pattern querying is Landmark [8] by Perng et al. It proposes a method to choose landmark points—points of the time series which best describe its pattern—and uses them in similarity checks. For our implementation we use time granularity aggregation, i.e. the mean, to represent the underlying time series. Piece-wise polynomial integration and regression has been used as a data reduction technique where it is used to reduce the frequency by aggregating nearby points. One such example is Best-Time [10] where the authors use piece-wise aggregation to reduce the frequency of original time series in to manageable frequency. For example if the frequency in original time series is 5 Hz and required frequency is 1 Hz, the deduction process aggregates 5 points using interpolation or regression method and present it with 1 point. In our aggregated mean time series each original segment is represented by its mean value. Thus an approximated version of the original time series is created. The number of points used for aggregation can be defined by the user. Using more points for the aggregation leads to a more compressed but also more coarse-grained model that can compromise accuracy. In our implementation we used 5, 10 and 20 points as aggregation levels which are similar to compressed levels in other types of models.

Discrete Fourier Transformation. Discrete Fourier Transformation (DFT) is a widely known technique to represent complex signals, using a collection of symmetric signals, where each signal is represented with two components. A mathematical representation of DFT decomposition can be described as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \tag{1}$$

where $x_0, ..., x_{N-1}$ are complex numbers. Evaluation of this equation has $O(N^2)$ time complexity that can be reduced to $O(N\log N)$ with Fast Fourier Transformation algorithms such as CooleyTukey or Prime-factor FFT.

A time series can be viewed as a signal when we consider it as a continuous signal rather than discrete values in discrete time points. Thus, we use FFT to analyze and represent our time series. When using DFT/FFT it returns a collection of components with the same number of points in the input time series. Not all these components are important as the latter part is a repetition of the first part in a mirror form. In order to obtain an approximation, a fewer number of FFT components is adequate.

Discrete Cosine Transformation. Similar to DFT, Discrete Cosine Transformation also decomposes a given signal to simpler components and uses these components to get an approximated signal. DCT is widely used in image and signal compression as it has a strong energy compression behavior. DCT-II is the most common compression technique and can be formalized as:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{2}$$

where $k = 0, ..., N-1$ and $x_0, ..., x_{N-1}$ are real numbers in contrast to the complex numbers in DFT/FFT.

The selection of DCT or DFT is based on user requirements. Model construction for DCT is more expensive than for DFT. In contrast DCT models have a smaller memory footprint compared to DFT models with the same number of components. As we suggest offline and online model construction, we evaluate both approaches in our experiments.

3.2 Query Computation over Models

In its current state, our implementation supports standard SQL aggregate queries like AVG, SUM and Histogram analysis. Because our query results are approximations, they significantly increase the run-time efficiency of query processing. As SUM queries are one of the most widely used aggregation query types, we first demonstrate how they can be answered with FFT and DCT models. Assume a query such as;

```
SELECT avg(sales)
FROM sales
WHERE date BETWEEN 01-01-2010 to 31-12-2013
USE MODEL DCT
ERROR WITHIN 10 %
```

After parsing the query, the DCT model category is selected to run the query and decides the model parameters using the given error boundaries. The number of sub-models which are used in the query depends on the *WHERE* predicate.

We denote that each model has its own way of calculating aggregation parameters. We begin by defining our models as $\Sigma(TS) = [\sigma(ts_1), \sigma(ts_1), ..., \sigma(ts_n)]$ where $\sigma(ts_i) = [c_1, c_2, c_3, ..., c_k]$ and k is the number of FFT/DCT components. Each $\sigma(ts_i)$ represents a single partition (size C) of the main time series and its index corresponds to the partition ID. The inherent feature of FFT and DCT transformation is the first component which is also known as the base component. This component is a summation of all points from the original signal. Thus calculating SUM—and also AVERAGE—queries can be performed as follows: We define Start Point (SP) and End Point(EP) of the queried time series, '%' as the modulo operator and '/' as the integer division. The reason for integer division is, the resulting value can directly used as the index to access the model. For simplicity we consider an FFT model but the same applies for DCT models too.

$$Q_{SUM} = \sum_{n=SP\%C}^{C} \frac{\sum_{i=0}^{C} \sigma(ts_{\lfloor SP/C \rfloor})[i] e^{-i2\pi k \frac{n}{C}}}{C} + \sum_{p=\lceil SP/C \rceil}^{\lfloor EP/C \rfloor} \sigma(ts_p)[1] + \sum_{n=1}^{EP\%C} \frac{\sum_{i=0}^{C} \sigma(ts_{\lceil EP/C \rceil})[i] e^{-i2\pi k \frac{n}{C}}}{C} \tag{3}$$

We consider $\sigma(ts_{\lfloor SP/C \rfloor})$ as a vector taken from the matrix representation of the larger model $\Sigma(TS)$ with its partition ID matching the raw ID $\lfloor SP/C \rfloor$. As shown here it is not necessary to approximate the full length of the queried time series but only the head and tail parts. This is due to the inherent feature of DCT and DFT transformation. In both DCT and DFT the inverse transformation from only the first component defines the base components which is the mean value of all the points being considered for transformation.

4 Evaluation

We are currently in the process of implementing the model query processor, thus our initial results are based on finding sufficient query statistics to support the model query processing. For example we want to evaluate when we should use the model query processor based on queried data length etc. We also store statistics such that given the query parameters, we can select different models. In order to show the efficiency and effectiveness of our approach we conduct extensive experiments with different time series, under different settings. We have implemented two models from the signal processing domain known as DFT (also known as FFT) and DCT to use with our time series. We also implemented an approximated aggregate model to be used with time series data.

Data: We have conducted our experiments with different datasets. We obtained NREL wind data set[1] where we extract data from 50 different wind turbines. We also use the UK Energy Demand data set which has a pretty nice seasonal pattern. Most of our results are based on the NREL wind dataset. As individual time series from the NREL wind dataset contains only around 157000 data points each with 10mins resolution. In order to derive sufficiently large time series, we extend the original time series by permutation with other time series in the dataset. The resulted time series have around 3140000 data points.

Experimental Setup: We used a standard personal computer with an Intel Core i5 processor, 8GB RAM with Windows 8.1 Operating System. We used R to implement our system. In order to implement standard query processing, which reads raw data and produce results, we use standard R functions SUM and AVERAGE. Thus, base results of our experiments are from standard R functions.

[1] http://www.nrel.gov/gis/data_wind.html

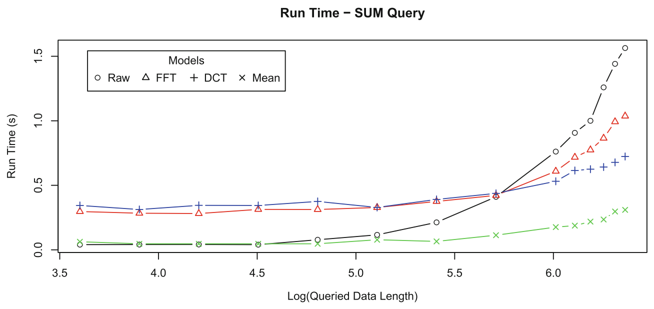


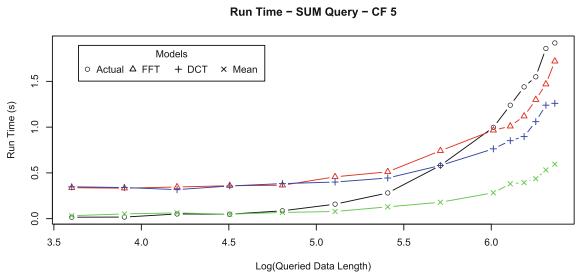
Fig. 3. Running time for different query lengths on different models and running time of the original raw data for Wind Data time series.

4.1 AVG and SUM Queries

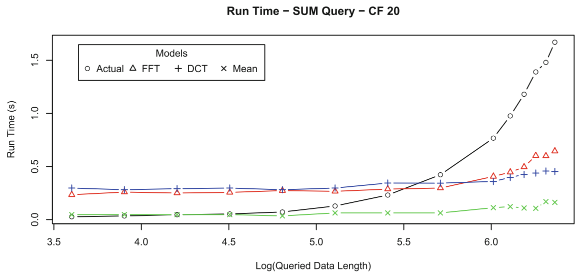
One of our query parameters is the query run time. When a query is being issued with time bounds it is necessary to select suitable models. Thus we conduct experiment on query run time for SUM query against different models and the results are depicted in Fig. 3. As depicted in Fig. 3, we observe a significant improvement in performance wrt. query run time for SUM query when using the model based approach. One reason for this improvement is that IO which is significantly lower for models than for raw data. We can justify from our results that models with low memory footprint have better run-time performance. In this experiment models are one order of magnitude smaller than the raw data. We maintain these statistics in our query statistics such that future queries can be answered using the optimal approach (Raw Data or Models). We notice a break-even point at around 5.7 on the x-axis such that if the queried data length is larger than $10^{5.7}$ then the model query processor should be used to answer the query. The selection of models depends on the individual model characteristics and specified query parameters such as error boundaries.

Our models are adjustable such that a more compressed representation decreases query run time but also increase the error. Thus we evaluate query run time against compression factor. The results are depicted in Fig. 4 where it is visible that run time decreases significantly when we increase the compression. Having these results in our statistics helps to decide which model(s) to use for a particular query depending on the query parameters. Comparing Figs. 4(a) and (b) we can see that the model with compression factor 20 is roughly 3–4 times faster than the models with compression factor of 5. From that we can devise, Run Time and the Compression Factor are linearly proportional.

Use of models for information querying has an inherent problem. The models are abstract representations of the underlying data. The results obtained from the model do not exactly match with the result obtained from the raw data. Thus, we perform experiments to evaluate the accuracy of the results with respect to the same query issued on the raw data and experiment's results are depicted in Fig. 5. It can be seen that the larger the queried data, the better the



(a) Compression Factor 5



(b) Compression Factor 20

Fig. 4. Comparison of query run time changes with respect to compression

accuracy. Using our models, we approximate the points in head and tail parts of the time series being queried. The body part of the time series is calculated directly from the models associated with that part. There is no approximation needed for that part.

We also noted statistics on accuracy changes when the compression factor differs. In the previous experiment we show that higher compression leads to models with a lower memory footprint that have a lower execution time during query processing. It is also necessary to store the accuracy of different compression level to better answer user queries. Figure 6 depicts accuracy changes for compression factors 5 and 20 where we see high error in more compressed models. With these statistics the user always has the ability to trade off between performance and accuracy.

In our proposed method, we provide the functionality to the user where he/she can specify the query parameters like time boundary and accuracy boundary. Thus, in our model pool we store different models to support such specific parameters. In some cases we create simpler models from original stored models. Due to this flexibility it is necessary to evaluate the result accuracy with respect to the model specification. We use the compression factor as the model complexity where less compression means the model is more detailed and high compression means model is less detailed, thus, provides a more abstract representation. The results are shown in Table 1.

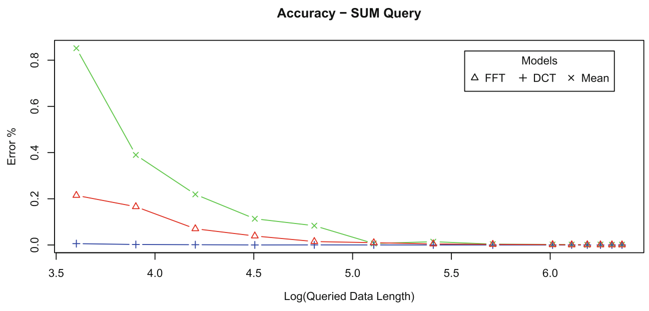
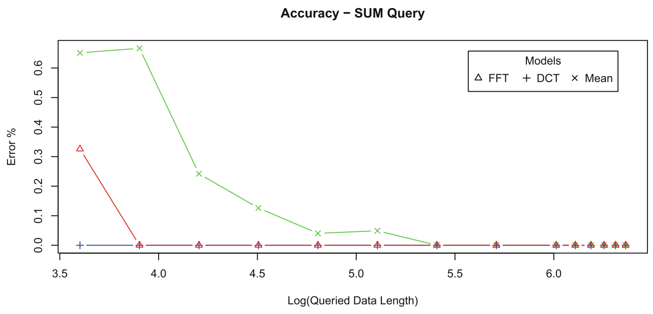
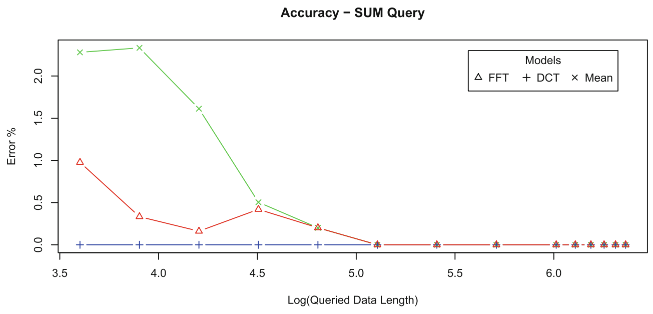


Fig. 5. Accuracy of SUM query on NREL wind dataset



(a) Compression Factor 5



(b) Compression Factor 20

Fig. 6. Comparison of query accuracy changes with respect to compression

Table 1. De-compression error of different models in different compression levels

		Model		
Compression	Measure	FFT	DCT	Aggregate
10	RMSE	4.269471	7.42458	15.00886
	MAPE	4.27367	8.048909	17.77352
5	RMSE	2.236411	3.552057	9.980734
	MAPE	1.851273	4.050521	10.51729
4	RMSE	1.879704	2.84323	6.990188
	MAPE	1.399919	3.025598	6.728125

4.2 Histogram Query

Histograms are a widely used analysis tool in the business domains and used as a metadata statistic in database and data warehouse domains for query optimization. A histogram shows the data distribution and we generate histogram from our models and analyze the accuracy of the generated histograms compare to histograms created using the original data. An example is shown in Fig. 7 where top left histogram represents original data, top right histogram represents reconstructed data using FFT model. Bottom left histogram and bottom right histogram represent DCT model and Mean aggregated model reconstructed histograms respectively. We want to analyze our decompression accuracy with respect to data distribution and similar to [9] we conduct Histogram Error Rate (HER) analysis where $HER(x) = \frac{\sum_{B}^{i=1} |H_i(x) - H_i(x)|}{\sum_{B}^{i=1} H_i(x)}$, defines error with respect to total bin differences in the histogram. The HER is 0.044, 0.063 and 0.152 for

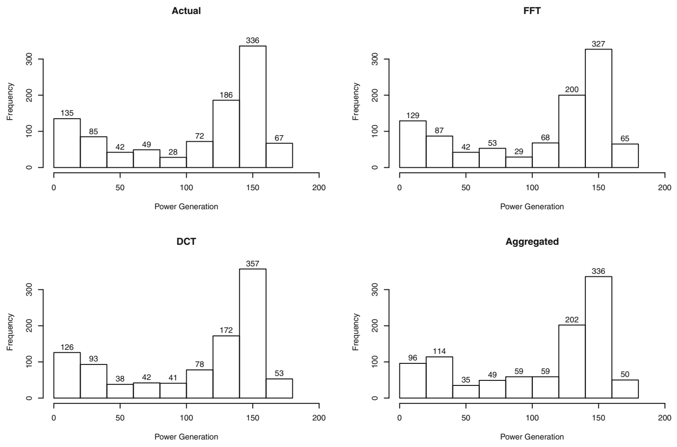


Fig. 7. Histogram representation of raw data and decompressed data from models in NREL Wind Dataset

FFT, DCT and Mean Aggregated models respectively. Mean aggregated model has three times higher error compared to FFT model. The compression factor in this experiment is 10 for all 3 models.

From our findings we can devise that different models have different characteristics. When we maintain all these models in our model pool together with their characteristics on different queries, it is necessary to select the optimal model to answer given query. Thus, our next step is to implement the model adviser which is responsible selecting the model to be used.

5 Related Work

Approximate query processing has been used in many application domains and new methods have been introduced to deal with general and application specific problems. Among these approaches, sampling based methods, histogram based methods and wavelet based methods are to name few.

BlinkDB [1] is a sampling based database system which provides approximate answers with time and error bounds. BlinkDB proposes stratified sampling which supports offline (samples are pre-calculated for the given query) as well as online sample creation when a new query is issued and cannot be answered by existing samples in the database. The system handles WHERE and GROUP BY queries and create samples accordingly. BlinkDB also maintains Error Latency Profile which are created with respect to sample size and their response time and relative error. BlinkDB cannot be used with time series data, which doesn't have multiple dimensions in contrast our models supports time series data. Interpreting a time series as a signal is a common technique in time series analysis. This approach gives the flexibility to use signal analysis methods to use on time series. One such approach is presented in [9] where they used FFT is used to compress original time series and represent it then through sampled Fourier components with residuals of original and approximated signal. They propose only histogram query analysis where we define SUM and AVERAGE queries over transformed models. We also introduce DCT models. Wavelet analysis (both continuous and discrete) is widely used in data analysis [2], in particular time series analysis. Discrete Wavelet transform-based time series and mining has been proposed in [3]. The proposed system can only handles multidimensional data and doesn't work well with time series data in comparison to our system. More recent work on approximate query over temporal data is presented in [7] which focuses on Frequency Domain Transformation which includes DFT, DCT and DWT. MauveDB [5] and its early implementation of model based data access [4] is another attempt to represent the underlying data through the models to provide a complete view of the data. MauveDB uses regression based models to represent the underlying data and does not provide compression. MauveDB presents a new abstraction level over the real data with errors and missing values to present a complete view to the users. This abstraction is built by using models represent to underlying data. As this layer provides transparency to the user, the user can still use SQL queries to derive information, and this information is provided by using models in the abstraction layer.

6 Conclusion and Future Work

Due to the large amount of data being generated by today's applications, it is necessary to introduce new information extraction methods as opposed to traditional database systems. In this paper we suggests a model based database system which underlying data is represented by models and queries are answered using models instead of using data. As a start we use models on time series data and show that it is possible to improve the performance by using models. Querying over proposed DFT (FFT), DCT and aggregation models have run-time efficiency gain upto 80 % as compared to querying over the original raw data and query results have high accuracy with less than 5 % of Mean Absolute Percentage Error (MAPE) with reference to exact answers.

As directions for future work, we suggest maintaining query statistics to determine when to use model based query processor and when to use data query processor. Our model-based database system implementation starts with time series representation as time series is the simplest form of data. Even though our model-based database system can apply only to few scenarios in time series domain, our system will have higher impact when representing data with many dimensions. In such cases models used for simple time series cannot be used as implemented now. Thus an extension of the current models to capture multiple dimensions as well as completely new set of models will be introduced in our future research.

Acknowledgment. This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate, Information Technologies for Business Intelligence - Doctoral College (IT4BI-DC).

References

1. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: Blinkdb: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013, pp. 29–42. ACM (2013)
2. Chakrabarti, K., Garofalakis, M.N., Rastogi, R., Shim, K.: Approximate query processing using wavelets. In: Proceedings of the 26th International Conference on Very Large Data Bases, VLDB 2000, pp. 111–122. Morgan Kaufmann Publishers Inc. (2000)
3. Chaovalit, P., Gangopadhyay, A., Karabatis, G., Chen, Z.: Discrete wavelet transform-based time series analysis and mining. ACM Comput. Surv. **43**(2), 6:1–6:37 (2011)
4. Deshpande, A., Guestrin, C., Madden, S.R., Hellerstein, J.M., Hong, W.: Model-driven data acquisition in sensor networks. In: Proceedings of the 30th International Conference on Very Large Data Bases - Volume 30, VLDB 2004, pp. 588–599. VLDB Endowment (2004)
5. Deshpande, A., Madden, S.: Mauvedb: supporting model-based user views in database systems. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD 2006, pp. 73–84. ACM (2006)

6. Khalefa, M.E., Fischer, U., Pedersen, T.B., Lehner, W.: Model-based integration of past and future in timetravel. Proc. VLDB Endow. **5**(12), 1974–1977 (2012)
7. Khurana, U., Parthasarathy, S., Turaga, D.S.: FAQ: a framework for fast approximate query processing on temporal data. In: Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine 2014, 24 August 2014, pp. 29–45 (2014)
8. Perng, C.-S., Wang, H., Zhang, S., Parker, D.: Landmarks: a new model for similarity-based pattern querying in time series databases. In: 2000 Proceedings of 16th International Conference on Data Engineering, pp. 33–42 (2000)
9. Reeves, G., Liu, J., Nath, S., Zhao, F.: Managing massive time series streams with multi-scale compressed trickles. Proc. VLDB Endow. **2**(1), 97–108 (2009)
10. Spiegel, S., Schultz, D., Albayrak, S.: BestTime: finding representatives in time series datasets. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) ECML PKDD 2014, Part III. LNCS, vol. 8726, pp. 477–480. Springer, Heidelberg (2014)