Towards Online Multi-Model Approximation of Time Series

Thanasis G. Papaioannou, Mehdi Riahi, and Karl Aberer
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland 1015
Email: firstname.lastname@epfl.ch

***Abstract*—The increasing use of sensor technology for various monitoring applications (e.g. air-pollution, traffic, climate-change, etc.) has led to an unprecedented volume of streaming data that has to be efficiently stored and retrieved. Real-time model-based data approximation and filtering is a common solution for reducing the storage (and communication) overhead. However, the selection of the most efficient model depends on the characteristics of the data stream, namely rate, burstiness, data range, etc., which cannot be always known a priori for mobile sensors and they can even be dynamic. In this paper, we investigate the innovative concept of efficiently combining multiple approximation models in real-time. Our approach dynamically adapts to the properties of the data stream and approximates each data segment with the most suitable model. As experimentally proved, our multi-model approximation approach always produces fewer or equal data segments than those of the best individual model. Finally, by employing both simple and more advanced indexing approaches for modeled data segments, we prove that multi-model data approximation is preferable to single model-based approximation in terms of response time for range-queries sent to big data tables.**

I. Introduction

Recent advances in sensor technology have enabled the availability of a multitude of (often privately-held) sensors. Embedded sensing functionality (e.g. noise, accelerometer, temperature, GPS, RFID etc.) is now included in mobile devices, such as phones, cars, buses, etc. Environmental and health-care applications based on community sensing in urban areas have been already envisioned, e.g. personalized carbon exposure and impact calculators, healthy lifestyle estimators, traffic monitoring, etc. Sensor mobility vastly increases coverage and provides new opportunities for emerging monitoring applications. At the same time, the large amount of these devices and the huge volume of raw monitored data pose new challenges for the sustainable storage and efficient retrieval of sensor data streams.

To this end, multiple regression and filtering techniques (referred to as *models*) have been proposed [1], [2], [3] for the online approximation of time series within a certain error norm (i.e. *lossy* compression). L_∞ is a common error norm that allows modeled data values to fall within a maximum error bound from the raw ones. These models exploit the correlations (e.g. with time or among data streams) exploited by time series to split data in pieces and approximate each segment with a certain mathematical function derived by the model. However, the potential varying burstiness (and possibly rate) of the data streams along time and the variable standard error introduced by the sensor mobility often result on limited effectiveness of a single model for approximating data within the prescribed error bound during a certain period. The same argument is also valid for other time series that may exploit variable burstiness in different time periods, e.g. stock prices during volatile or non-volatile market periods.

In this paper, we propose the innovative concept of combining multiple statistical models for approximating time series. The intuition behind this approach is that different data periods of a sensor stream can be better approximated by different models, thus resulting, overall, in fewer and longer segments. This is because of the greater flexibility offered by the different alternative models. We propose our multi-model longest-fit algorithm and prove its correctness for approximating the data stream within the specified error bound. By an extensive series of experiments with both real and artificial data traces, we prove that our approach always produces fewer or equal segments than any of its constituent models employed individually. Moreover, when linear models are employed, we experimentally show that our approach can achieve significant compression improvement over any constituent linear approximation scheme. We define an appropriate storage scheme and adopt an efficient indexing approach for intervals, referred to as RI-Tree [4]. As experimentally proved both for RI-tree and simple indexing schemes, multi-model compression algorithm results into significantly lower response times than individual models for value range and point queries. Our approach is fully implemented and can serve as a framework for efficiently combining arbitrary online approximation schemes for time series.

The remainder of this paper is organized as follows: In Section 2, we discuss further motivation for our work. In Section 3, we compare our approach to the related work and emphasize on the innovative characteristics of our approach. In Section 4, we introduce our multi-model approximation algorithm. In Section 5, we define the storage schema for our approach. In Section 6, we describe how the approximated time series can be efficiently indexed and queried. In Section 6, we present our experimental results that prove the effectiveness of our approach. Finally, in Section 7, we conclude our work.

II. Motivation

The wide availability of sensor technology has enabled a new era of emerging interesting applications in different contexts, e.g. environmental conservation, health-care monitoring, surveillance, safety, air-pollution monitoring, personalized carbon-impact assessment, etc. However, these applications continuously produce data streams that may be correlated, erroneous or incomplete. Moreover, the volume of this data may make them unmanageable in the long-run, if they are not properly stored. Probabilistic models [3], [5], [6], [7] have been proposed for data cleaning and for deducing hidden variables. However, they are not appropriate for compressing the data in storage, as they estimate, store and provide probability distributions (usually taking more storage space than the original data), instead of actual values. In Section III, we reviewed multiple statistical (e.g. regression) and other filtering techniques that have been proposed for data compression in storage and in communication. These models built online and store in database tables the approximated data segments instead of the raw stream values. Thus, in general, the *fewer* the segments that approximate the complete data stream, or alternatively the higher the average segment length, the higher the compression that can be achieved. However, the achievable compression ratio also depends on the complexity of the model that determines the necessary storage space for the representation of each stream data segment in the database.

Based on their generation algorithm, different online approximation models exploit the correlations of the data in a different way. For example, PMC-MidRange (or simply *MidRange*) piece-wise constant approximation is expected to approximate a *longer* data segment of a data stream oscillating across a certain constant value within the error bound, than a linear piece-wise approximation technique, such as *Linear* or *Swing* filters; the latter models require that the data stream values consistently follow a certain direction. This case is illustrated in Fig. 1, where a small period of a raw data stream produced by a temperature sensor is approximated by MidRange and Swing models. Only the edge points of the linear segments produced by each model are plotted in Fig. 1. As depicted therein, in the beginning of the stream period, Swing filter approximates with 1 linear segment a data stream subset of 17 raw values, while, for the same raw data, MidRange approximation model needs 3 linear segments. The situation is reversed at the end of the stream period, where 24 raw data values are approximated with 1 linear segment by MidRange, as compared to 3 linear data segments constructed by Swing for the same data. Another example is that, other stream trends in certain periods that involve multiple direction changes or periodicity, such as parabolic or sinoid, could be better approximated by 2nd-degree or 5th-degree polynomials respectively. Other properties of the data stream that can be differently exploited in different periods by the various approximation models, described in Section III, involve rate variability, oscillation length (i.e. burstiness), direction of trend, rate of trend change, the approximation error bound, etc.

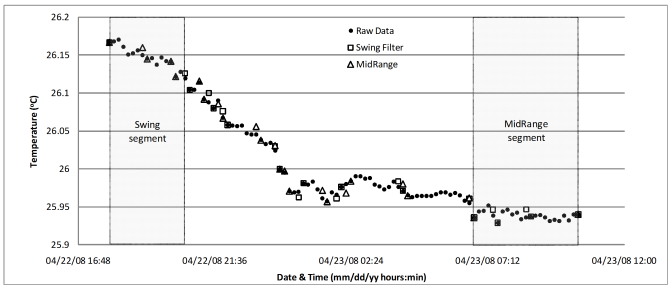

Fig. 1. Different time segments of the data stream are better approximated by different models.

III. Related Work

There is significant work in the literature related to model-based data compression. Among several different time series approximation techniques, piecewise linear approximation has been widely used [8]. Linear filter is a simple piecewise linear approximation technique in which the data points are approximated by a line connecting the first and second point of the segment. When the new point cannot be approximated by this line in the specified error bound, a new segment is started [1].

In [1], two new piece-wise linear approximation models were proposed, namely *Swing* and *Slide* and they were compared to previous linear models, such as cache and linear filters. Cache filter approximates data within a segment with a constant value, which can be the first value of the segment, the mean or the median (referred to as Poor Man's Compression-MidRange (PMC-MR) [9]).

In [10], a framework for partitioning and approximating raw data segments based on a user specified mathematical function or model is introduced. Data segments are stored in a special Function Table, coupling the raw data. Instead, our approach focuses on selecting an efficient collection of models that transparently approximate different data segments.

In [11], the piecewise linear approximation algorithms are categorized in three groups, sliding windows, top-down, and bottom-up. Among these three groups only the sliding windows approaches can be used online. The other two approaches perform better than sliding windows, but they need to scan all the data, hence they cannot be used for approximating streaming data. Based on this observation, the authors propose a new algorithm which combines the online property of sliding windows and performance of the bottom-up algorithm. While this approach can be employed by our multi-modeling technique, it needs a predefined buffer length. If the buffer is small then it may produce many small segments and if it is large, it will delay outputting the approximation of the data stream.

In [12], a general technique is introduced to reduce the space complexity of the offline and online optimal and approximate synopsis construction algorithms. The V-optimal histogram [13] for synopsis construction tries to find the best piecewise

constant approximation of n values with at most B pieces, while the sum of the square errors between the actual and approximated values is minimized. Instead, we try to maximize the compression ratio by approximating the data in real-time with multiple models while a predefined error bound is met.

Several approaches [3], [5], [6], [7] employ predictive probabilistic modeling based approaches for compressing the data that has to be communicated in wireless sensor networks. Deshpande et al. [14] also exploit spatial correlations among different data sources for compression. In the same context, but for optimizing storage space, the approach in [2] dynamically identifies and exploits correlations among different data streams and then jointly compresses them within an error bound employing a polynomial time approximation scheme. Our approach applies multiple micro-models per sensor stream and it can also exploit spatial correlations among different data streams or sensor readings from the same location on different attributes, e.g. humidity and temperature from the same sensor. Also, [15] defines a complete framework for scientific sensor data processing focusing on fast visualization of scientific models based on random walk sampling.

Also, [16] introduces a framework for defining model-based views of raw data based on statistical models that are of interest, such as linear regression and interpolation. Regression models aim to predict a dependent variable (e.g. next sensor measurement) based on the values of a set of independent variables (e.g. previous sensor measurements) and some calculated coefficients. The goal of regression models is to find the optimal weights that minimize some error metric given the set of observations and the expected values, such as the root mean square error, the standard deviation, the mean absolute error etc. On the other hand, interpolation models are a natural way to fill missing values. The approach in [16] does not focus on compression and it can be considered as complementary to our approach by employing approximated streaming data instead of raw data to construct model-based views. In [17], the work of [16] is extended providing a framework to define and maintain views based on dynamic probabilistic models [18], [19]. In [17], particles are employed for representing dynamic probabilistic models, which are weighted samples over conditional probability distributions (CPD) that are learned based on data using Maximum Likelihood Estimation (MLE). The weights on particles are updated using the CPD of the observed values.

Finally, Babu et al. proposed in [20] a heuristic approach for selecting the classification and regression tree (CaRT) models that semantically compress columns of huge data tables based on value correlations within some error bounds in accuracy. However, this approach is rather offline, as the whole data table should be available as input to the compression algorithm which is rather costly, as opposed to our approach which compresses the data in real-time and thus better suited to streaming data.

IV. The Basic Algorithm

Online piece-wise approximation algorithms seek to find the parameters of a certain mathematical function, so as to fit the raw values of a segment of the data stream within a maximum error bound. When a raw data value of the stream cannot be approximated within the error bound by a specific instantiation of the "fitting" function of the model, then a new data segment is initiated; within the new data segment, a new instantiation of the fitting function has to be found and employed for data approximation. Each approximation model has to employ a fixed number of initial raw values in a data segment, in order to find the instantiation of its fitting function for this segment; e.g., 1 value for the cache filter, 2 values for the linear filter, 3 values for 2nd-degree polynomial regression, etc. We refer to this model requirement as *initialization*. The raw data stream values of the segment necessarily fit into the model function during its initialization, while the minimum initialization length that may be required by a model is 1 value.

In our approach, a collection of models are jointly employed for approximating the data stream. Each data segment is approximated by the most effective model instantiation for that segment. The model effectiveness for a segment is determined based on the *segment length* (i.e. fitting period) in terms of raw data values that can be approximated by the same instance of the model and its achievable *compression ratio* for this segment.

More formally, we want to construct a multi-segment mathematical formula. For simplicity, we consider the approximation of a single attribute of the stream that exploits a temporal correlation.

$$H(t) = \begin{cases} h_1(t)\,, & t \in [t_0,\ t_1) \\ h_2(t)\,, & t \in [t_1,\ t_2) \\ & \dots \\ h_N(t)\,, & t \in [t_{N-1},\ t_N]\,, \end{cases} \tag{1}$$

where h_i is a certain instantiation of the formula $f_m(t)$ of the model m that achieves the highest compression ratio for the data in the time period $[t_{i-1},\ t_i]$.

We want to select the model m that approximates each segment i, so that the total number N of segments is minimized. Minimizing the number of segments necessarily achieves the best compression ratio, since the model selected at each segment is the cheapest in terms of storage requirements.

For finding the offline optimal solution to the problem, all possible combinations of models should be enumerated, which is a NP-complete problem. To this end, we propose a greedy algorithm (described in Algorithm 1) for selecting for each segment the model that i) maximizes its length (i.e. approximates the largest number of raw values), and ii) it is the cheapest to be stored. The algorithm achieves this as follows: Consider a set M of models that jointly approximate a certain data segment of the data stream S. Each raw data item $< t, v >$, i.e. with value v at time t, is examined by each of the initialized models for this data segment whether it falls ("hit") or not ("miss") within the

error bound ϵ from the estimated data value by the model at time t, i.e. whether $|f_m(t) - v| < \epsilon$ or not for a model m with an instantiated function f_m. All uninitialized models succeed into approximating the raw data item $< t, v >$ in this segment by default. We calculate the compression ratio r_m for each missing model m and we exclude m from the models that are further examined against the aforementioned *hitting condition* for this segment. We repeat examining the hitting condition for all subsequent raw data items of the stream, until all models "miss". At this point, the model m^* with the highest compression ratio r_{m*} is dumped into the database to approximate the data segment until the time t_{m*} that it missed. Afterwards, the data stream is retracted to time t_{m*}, the approximation formulas of all models are cleared and all models are considered for the approximation of the next data segment according to the aforementioned procedure. If at the time that all models miss, there are several missed models with the same highest compression ratio, then the one with minimum root mean squared error is selected to be dumped in the database.

The formulation (1) applies to the approximation of the data stream with *connected* segments. In this case, the approximation of a new segment starts from the ending time of the last segment. In case that a data stream is approximated by *disconnected* segments, the approximation of a new segment starts at the time of the "miss" for fitting a raw data stream value.

The optimal offline combination of arbitrary models could be found by Dijkstra algorithm for finding the shortest path in communication networks as follows. The nodes are the data items of the stream, while the models are potential outgoing links at each node. Each model leads to a different ending time for a segment and it has a different storage cost that is used as the weight of the link. Each node is connected to its subsequent node in the stream by a link annotated with the cost of storing a linear segment. We want to find the path across the data items from the starting time of the stream until now, which is expected to achieve the highest compression ratio, i.e. the minimum storage cost. Overall, this algorithm would have complexity $O(N|M| + N \log N)$.

The uncertainty of our greedy algorithm lies on the fact that the selection of a certain model that maximizes the compression ratio for approximating a segment, may lead to subsequent increase of the number of segments that are required to approximate the data stream. However, as experimentally proved in Section VII, our greedy algorithm for multi-model approximation always produces fewer or equal segments to the ones produced by the most efficient of the models when individually employed for approximating the data stream.

A. Correctness

Theorem 4.1: The approximate data stream produced by our multi-model fitting algorithm always satisfies the pre-specified error bound.

Algorithm 1 Multi-model data approximation

Require: Set M of models, stream $S = \{< t_i, X_i >\}$
  $F \leftarrow M$
  $F^* \leftarrow \emptyset$
  while $|S| > 0$ **do**
    $< t, v > \leftarrow fetch(S)$
    if $\forall m \in M$, $is\_initialized(m) =$ **false then**
      $t_0 = t$
    end if
    for all $m \in F$ **do**
      if $is\_initialized(m) =$ **false then**
        $initialize(m, < t, v >)$
        continue
      end if
      if $|v - f_m(t)| > \epsilon$ **then**
        $F = F \setminus \{m\}$
        $F^* = F^* \cup \{m\}$
      else
        $t_m \leftarrow t$
      end if
      {check if all models are dropped}
      if $F = \emptyset$ **then**
        $f_{m*} \leftarrow f_m$ s.t. $m^* \in F^*$ and r_{m*} is maximum
        $dump(t_0, t_{m*}, f_{m*})$
        $F \leftarrow M$
        $F^* \leftarrow \emptyset$
        for all $m \in F$ **do**
          $clear(m)$
        end for
        $retract(< t^*_m, v^* >, S)$
      end if
    end for
  end while

Proof: Assume that a certain raw data item is not approximated within the error bound. Then, there should be a data segment that includes this data item approximated by a certain model that violates the error bound within the segment. However, according to Algorithm 1, a violation of the error bound would lead to the automatic exclusion of the missing model from the considered ones for this data segment. If all approximating models concurrently violated the error bound for this particular raw data item, then the previous data segment would be dumped to the database, while the raw data item would belong to a subsequent segment. Then, if again no model could approximate this raw data item in the new segment, then the raw data item itself would be dumped to the database, i.e. as in lossless approximation. Therefore, there cannot be a raw data item that is not approximated by our algorithm within the pre-specified error bound. ■

V. Storing

There are multiple alternatives for model materialization in the databases, based on the purpose of the approximation. For example, if models are employed for constructing data views,

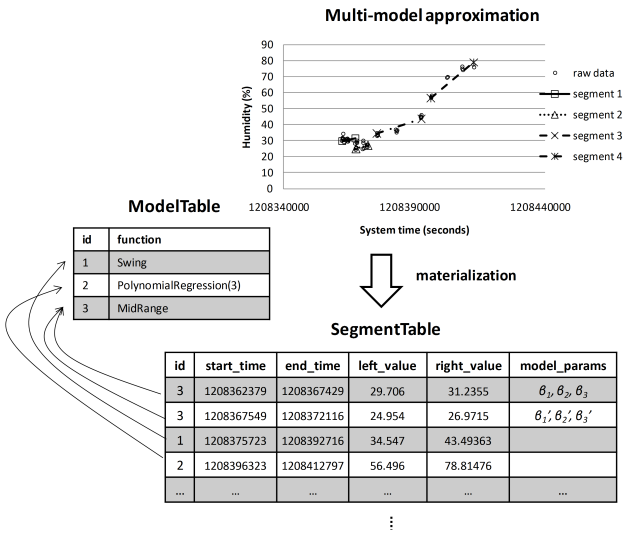

Fig. 2. The database schema for multi-model materialization.

then the view data have to be either materialized and updated in real-time or re-calculated on-demand based on raw data. The former approach may involve huge storage requirements, while the latter may introduce huge latency. Since we are employing the approximation models for compression purposes, only the approximated data segments are stored in the database, instead of the raw values of the data stream. A generic database schema for multi-model approximation consists of one table (*SegmentTable*) for storing the data segments, and a second table (*ModelTable*) for storing the model functions, as depicted in Figure 2.

A tuple of the *SegmentTable* contains the approximation data for a segment in the period [$start\_time$, end_time]. The attribute id stands for identification of the model that is used in the segment. The primary key in the *SegmentTable* is the $start\_time$, while in the *ModelTable* is the id.

When both linear and non-linear models are employed for the approximation, $left\_value$ is the lowest raw value encountered in the segment and $right\_value$ is the highest raw value encountered in the segment. In this case, $start\_time$, end_time, $left\_value$, $right\_value$ define a rectangular bucket that contains the values of this segment, which is employed for indexing purposes, as explained in Section VI. Also, the attribute $model\_params$ stores the parameters of the instance of the model id that approximates this segment, e.g. the regression coefficients for a regression model. The attribute $model\_params$ has variable length (VARCHAR or VARBINARY data types in SQL) and it stores the concatenation of the parameters or their compressed representation (by means of standard compression techniques, such as Deflate(gzip), bzip2, LZMA(7zip), etc.). Each tuple in the $ModelTable$ corresponds to a model with a particular id and a certain $function$. The attribute $function$ represents the name of the model and it corresponds to the name of two user defined functions (UDFs) stored in the database: i) one that implements the mathematical formula of the model, and ii) one that implements the inverse mathematical formula of the model, if any. Both are employed for answering value-based queries, and specifically the latter for equation solving, while the former for value regeneration in fixed time steps, referred to as *gridding*.

When only linear models are employed, the attributes $left\_value$, $right\_value$ are the edge values of a linear segment, and thus the attributes id, $model\_params$ can be omitted from the storage schema. Thus, an individual linear segment is less expensive in storage than a non-linear one.

Also, if the multi-model approximation algorithm is configured to produce connected segments, then the end_time attribute could be omitted from the segment tuple. However, this option would deteriorate the efficiency of the storage scheme for answering time range and point queries, as proper indexing of data segments would be no longer possible.

VI. Indexing and Querying

An efficient storage scheme should not only compress the size of the data on disk, but also allow for efficient query answering. Here, we discuss indexing solutions for efficient data retrieval from our storage scheme for time-based and value-based queries.

A. Time-based queries

In order to be able to efficiently retrieve the approximate stream values in particular time points or intervals, we have to build two conventional $B+$-tree indexes on $SegmentTable$: one on attribute $start\_time$ (it already exists since $start\_time$ is a primary key) and a second one on attribute end_time. This way, both linear and non-linear models approximate the data stream, the data segments of interest can be efficiently retrieved and the ids of their respective approximation models are employed for finding the model names from the $ModelTable$. The user defined functions (UDFs) that implement the mathematical formulas of the models are then employed for data gridding and answering the queries. If only linear models are employed in multi-model approximation, then the segments of interest already contain the information to derive the mathematical formula that approximates the segment (i.e. a line).

B. Value-based Queries

Here, we discuss on how we can efficiently answer valued-based range and point queries. To this end, we first describe the appropriate use of conventional $B+$-tree indexes and we subsequently discuss the adoption of a more complex indexing approach, referred to as Relational Interval Tree (RI-tree) [4]. While our proposed methods based on RI-tree can efficiently answer the full range of queries that involve predicates of the SQL:1999 standard datatype PERIOD, namely $precedes$, $succeeds$, $meets$, $equals$, $overlaps$, $contains$ and $during$ [4], here, we focus only in the $overlap$ case.

1) Conventional indexes: An $overlap$ query basically searches for all the intervals that overlap with the given interval $[lower, upper]$, i.e. their intersection with the query interval is not empty. The overlapping intervals can be found by using the following SQL query.

```
SELECT id FROM SegmentTable st
   WHERE st.left_value <= :upper AND
       st.right_value >= :lower;
```

In order to efficiently answer this query we can build a composite index on the attributes $(left\_value, right\_value)$. For answering point queries, we can represent the query point as an interval with the same endpoints and then use the above query. After the relevant model ids were retrieved, their corresponding model names can be extracted from the $ModelTable$. The user defined functions for each model are consulted and the final results are obtained either mathematically from the inverse of the function or by gridding. If only the linear models are used in multi-model approximation, then the results can be directly obtained from the lines that represent segments.

2) RI-Tree: The Relational Interval Tree (RI-tree) [4] is a method to efficiently support intersection queries on intervals stored in databases. RI-tree employs built-in indexes without augmenting any internal data structure in the database, hence it can be used in any relational database management system (RDBMS).

The RI-tree technique maintains (virtually) a backbone tree structure, a balanced binary tree which assigns intervals to its nodes. We adopt the basic version of RI-tree, where the root node is equal to 2^{h-1} and $2^{h-1} <$ data range $< 2^h$. The left child of a node x has value $x - step/2$ and the right one has value $x + step/2$, where $step$ is initially 2^{h-2} and it divided by 2 at each descend. As the tree is descended, an interval $[l, u]$ is assigned to the first node that falls within the interval, for which is referred to as its *fork node*. The RI-tree requires one table $(node, lower, upper, id)$ for the intervals and two composite indexes, namely *lowerIndex* on $(node, lower)$ and *upperIndex* on $(node, upper)$. A new interval is assigned to its corresponding *fork node* in the virtual tree by descending the tree. The fork node is stored at the $node$ attribute for this interval. As the tree is repeatedly traversed for interval insertions, a variable $minstep$ stores the minimum value of $step$. According to the RI-tree method, during interval searches, the tree is further descended only if $step > minstep$. The original version of RI-tree was designed for storing integer values. We modify the basic RI-tree method for storing real values as well by allowing $step$ to take values arbitrarily lower than 1. Thus, we can store intervals of arbitrarily small decimal range, while the height of the RI-tree becomes $h - \log(minstep)$.

Answering an intersection query in RI-tree technique consists of two steps. First, the tree is searched and the nodes at which the potentially overlapping intervals are registered are collected in two intermediate relations *leftNodes(min, max)* and *rightNodes(node)*. Second, a SQL query is issued on the interval schema and the intermediate relations to find all the overlapping intervals. For a query interval $[lower, upper]$, the primary structure is descended as follows. First, we descend from the root node down to the node preceding the fork node for the interval. For each node x along that path, if $x < lower$, then its intervals $[l, u]$ intersect $[lower, upper]$ exactly when $lower < u$. In this case, a tuple (x, x) is inserted into the *leftNodes* relation. On the other hand, if $x > upper$ then the interval $[l, u]$ registered at x intersects $[lower, upper]$ exactly when $upper > l$. In this case, a tuple (x) is added to the relation *rightNodes*. In the next step, we descend from the fork node down to the closest node to $lower$. Along this path, if $x < lower$, then a tuple (x, x) is added to the *leftNodes*. Last, we descend from the fork node down to the closest node to $upper$. Along this path, if $x > upper$ then a tuple (x) is added to the *rightNodes*. After inserting the query interval $(lower, upper)$ itself to the *leftNodes*, the following SQL query is then issued to report all the intervals that intersect the query interval by only employing *lowerIndex* and *upperIndex* indexes.

```
SELECT id FROM Intervals i, leftNodes left
WHERE i.node BETWEEN left.min AND left.max
AND i.upper >= :lower
UNION ALL
SELECT id FROM Intervals i, rightNodes right
WHERE i.node = right.node AND i.lower <= :upper;
```

The RI-tree requires $O(n/b)$ disk blocks of size b to store n intervals, $O(\log_b n)$ operations for insertion or deletion, and $O((h - \log(minstep)) \cdot \log_b n + r/b)$ I/Os for an intersection query that produces r results.

VII. Evaluation

A. Experimental setup

We fully implemented our multi-model approximation algorithm in Java and the proposed database schema both in Oracle 11g and in MySQL 5.1. Moreover, all proposed indexing approaches were implemented in both databases. For the RI-tree method, two main user defined functions (UDFs) were implemented, namely $insertInterval()$ and $queryIntervals()$, for inserting an interval into the index and for sending value-based range queries respectively. Our experiments were run on a (Core2Duo 2.5GHz CPU, 4GB) machine. As we obtained similar results from both database implementations, we will only present our results with Oracle 11g (default parameters, 580MB memory).

B. Compression

In our experiments, both real and synthetic data sets were employed. As real data sets, we used measurements for various environmental parameters (air temperature, humidity, wind direction) collected from existing sensor deployments in the Swiss Alps by the Swiss Experiment project (www.swiss-experiment.ch) and sea surface temperature sensor data from the TAO project (www.pmel.noaa.gov/tao/). Air temperature, sea surface and humidity time series exploit smooth statistical behavior (due to their inherent physical laws), while the wind direction data set is highly *bursty* and quite unpredictable in

nature. We also used two synthetic data sets. As a synthetic data set, we generated *Lorenz* time series. The Lorenz attractor is a three-dimensional structure corresponding to the long-term behavior of a chaotic flow [21]. We employed as Prantl number $\sigma = 10$, as Rayleigh number $\rho = 28$, and as physical proportion $\beta = \frac{8}{3}$, in the ordinary differential equations of the Lorenz attractor. For the Lorenz synthetic data set, we calculated 10000 Lorenz samples with a step width of 10^{-2} and employed the x-coordinate of the attractor.

All data sets were approximated within 3 different maximum error bounds: 3.16%, 5%, and 10% of the data range. We implemented multiple models including Swing (SW), MidRange (MR), Linear Filter (LF), Linear Regression (LR), Least Squares Line (LS), Constant Filter (CF), and Chebyshev Polynomial with different degrees (referred to as *Cheb* in figures with the degree specified in parenthesis).

a) Connected vs. Disconnected segments and Optimality: We first assess the effectiveness of our multi-model approximation algorithm for compressing a data stream with *connected* or *disconnected* segments. Recall from Section V that, when connected segments are employed, each data segment could be represented by one attribute less (i.e. end_time is redundant), as compared to the disconnected segments. However, these space savings would come at the cost of inefficient data retrieval in the case of time-based range queries. Thus, the omission of end_time can be exploited only for data communication compression. On the other hand, disconnected segments offer more flexibility to the models for approximating data, as they do not have to start the approximation of a new segment from the ending time of the previous one. In this experiment, we approximate the data combining six different linear models, namely Swing (SW), MidRange (MR), Linear Filter (LF), Linear Regression (LR), Least Squares Line (LS) and Constant Filter (CF). As depicted in Figure 3, our algorithm with disconnected segments achieves better storage compression than with connected ones both for sea temperature and for wind direction data sets, and for all the different maximum error bounds considered. The achieved compression improvement increases for more bursty data sets, such as wind direction, as shown in Figure 4. Similar results were obtained for all different data sets considered and for various combinations of both linear and non-linear models. Therefore, disconnected segments are employed by the models for data approximation in the rest of this paper.

b) Multiple linear models: We now assess the compression effectiveness of our multi-model approximation as compared to the compression achieved by its constituent linear models when individually applied to the data stream. Combining linear models is interesting, because they are computationally very efficient, very cheap to store as discussed in Section V, and they achieve comparable effectiveness to their more complex counterparts. As depicted in Figures 5 and 7, our multi-model algorithm achieves better compression ratio, than any individual linear model, both for smooth (humidity) and bursty (wind direction) real time series respectively. However, the more bursty the data set, the lower the achievable compression ratio. Also, as shown in Figures 6 and 8, our approach indeed produces up to 80% fewer segments than any of the individual models, according to its design objective. The better compression effectiveness of the multi-model approximation algorithm over the individual linear models remains stable for the synthetic data sets, e.g. as depicted in Figures 9 and 10 for the Lorenz data set. Figure 11 depicts how the multi-model approximation selects different individual models.

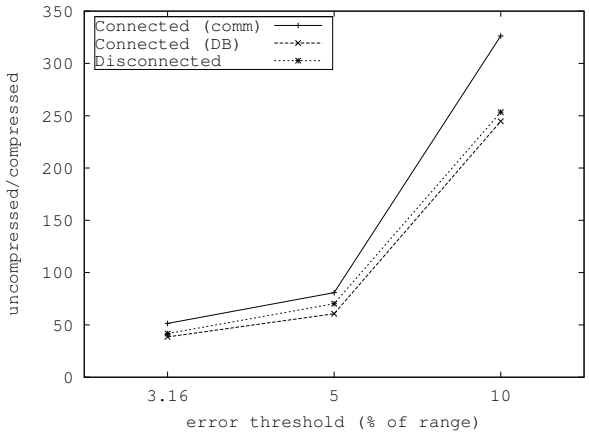

Fig. 3. Compression ratios achieved by generating connected compared to disconnected segments (Ocean temperature dataset). If we want to store the connected segments in a database using the proposed schema, the compression ratio degrades as we need to store both start and end timestamps.

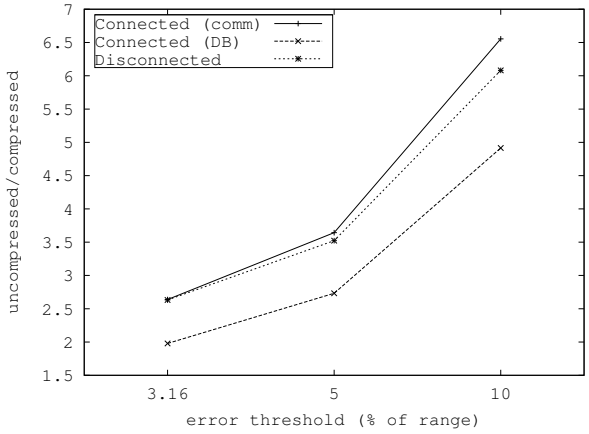

Fig. 4. Compression ratios achieved by generating connected compared to disconnected segments (wind direction dataset). If we want to store the connected segments in a database using the proposed schema, the compression ratio degrades as we need to store both start and end timestamps.

c) Combination of Linear and Non-Lineal Models: We finally assess the compression effectiveness of our multi-model approximation algorithm when both linear and non-linear models are combined together. Non-linear models may be more effective to approximate complex data trends of time series, but their segments are also more costly to be stored. As shown in Figure 12, the combination of multiple linear and non-linear models achieves the highest compression ratio for approximating the Lorenz synthetic data set. More

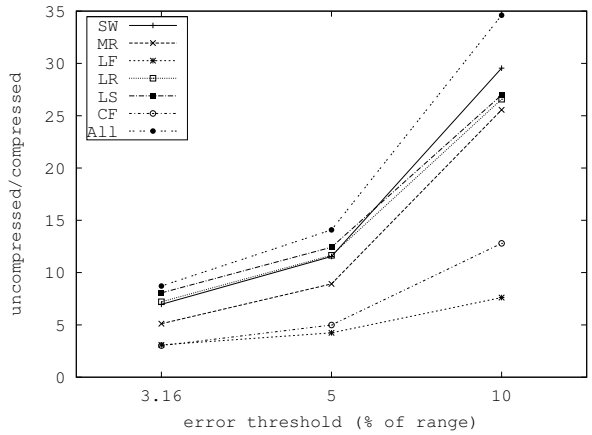

Fig. 5. Compression Ratio for Humidity dataset

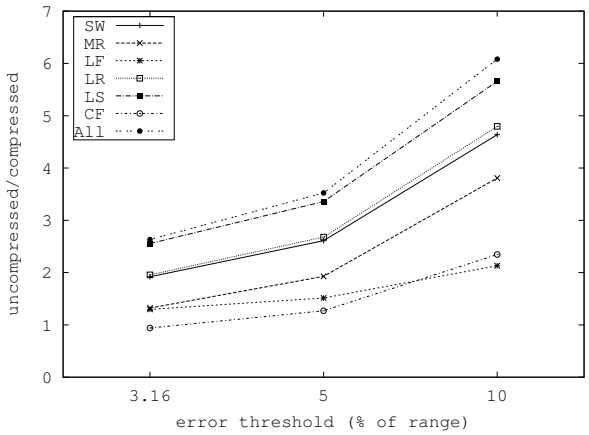

Fig. 7. Compression Ratio for Wind dataset

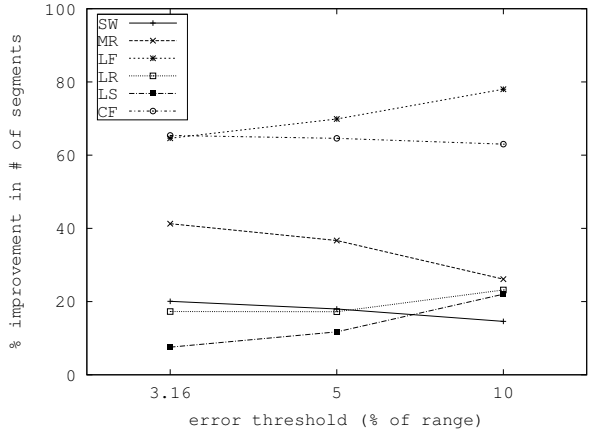

Fig. 6. Improvements in the number of segments for Humidity dataset

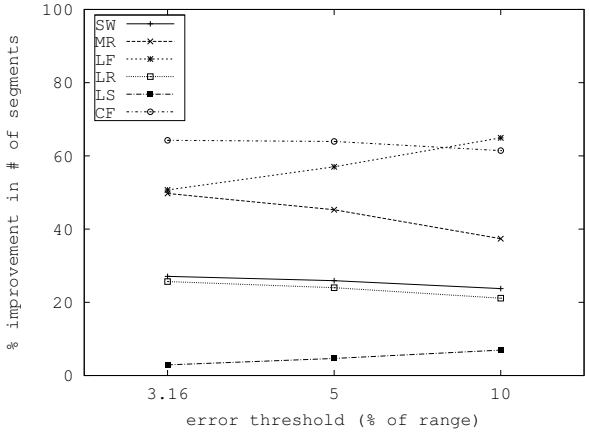

Fig. 8. Improvements in the number of segments for Wind dataset

interestingly, combining linear and non-linear models for approximating this data set has better compression effectiveness than the combination of only the subset of linear models. In this case, our greedy algorithm achieved to exploit the most of the benefits and avoid most of the weakness of its constituent models. Almost the same result is achieved for the sea surface time series in Figure 13. The contradiction here is that for maximum error bound 10%, the combination of the subset of linear models achieves slightly better compression ratio than the combination of all models. Still, this result only means that our multi-model combination algorithm is very efficient into finding a combination of models that achieves higher compression than the individual ones, however, it is not guaranteed to find the *optimum* model combination, as expected.

C. Data Retrieval

Here, we evaluate the effectiveness of multi-model approximation algorithm for data retrieval when the indexing solution of Section VI are employed. It is expected that as our algorithm approximates the data series with fewer data segments than its constituent models when individually used, it also results into lower index sizes and consequently better query response times.

In the absence of very long real data set to be approximated, we generate a *synthetic-linear* data set, as follows: The value x_i of point i is calculated as $x_{i-1} + s_i \cdot r_i$, where s_i can take -1 or 1 each with the probability of 0.5 and r_i is the amount of decrease or increase which is selected from the uniform distribution $U(0, 100)$. Each value is also forced to be in the interval $[0, 100]$. These data series were approximated by an efficient linear model, namely Swing (SW), and by our algorithm combining Swing (SW), Linear Filter (LF), Constant Filter (CF), Least Squares Line (LS), Linear Regression (LR), and MidRange (MR) models within a maximum error bound of 7.5% of the data range. We generated five long enough synthetic-linear data series, so that the Swing model produces 10k, 100k, 1m, 10m, and 20m segments respectively. The data segments produced both by the multi-model algorithm and by Swing are stored in the database and indexed by conventional and RI-tree indexes, as described in Section VI. Next, we generated a series of random queries for intervals of various fixed lengths, specifically of 2, 4, 8 and 16, and we sent them to the database. The length of the queried interval determines

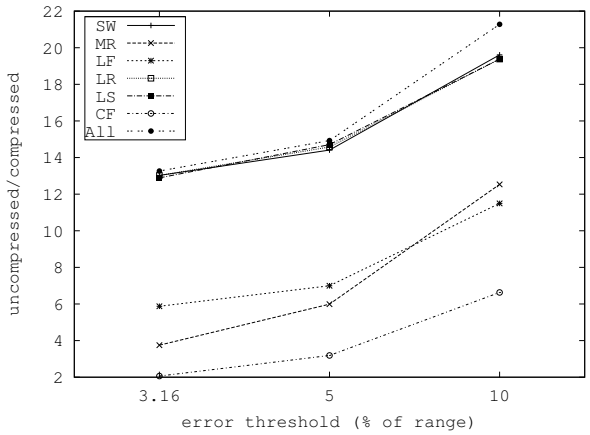

Fig. 9. Compression Ratio for Lorenz dataset

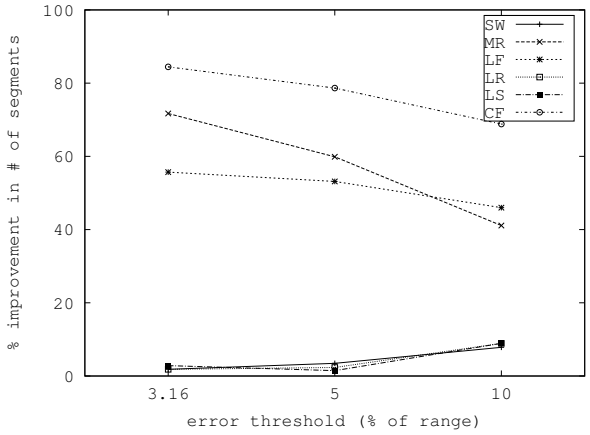

Fig. 10. Improvements in the number of segments for Lorenz dataset

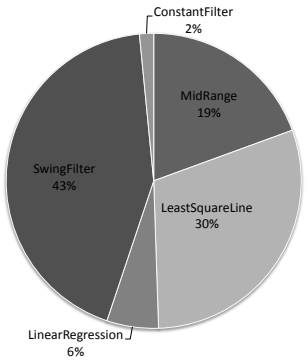

Fig. 11. The distribution of selected individual models for Humidity dataset experiment with %10 error threshold

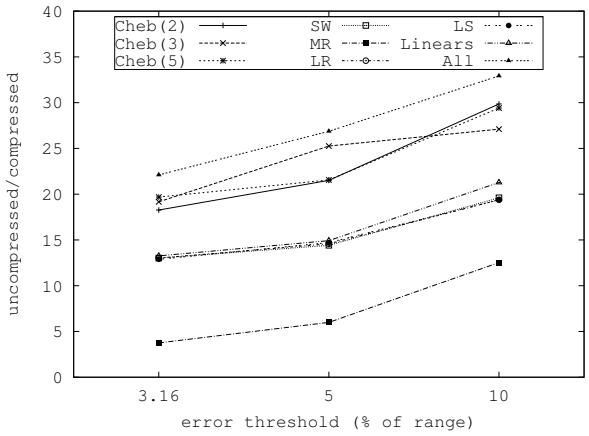

Fig. 12. The compression achieved by combining linear and polynomial models compared to combining only linear models (Lorenz dataset).

its selectivity. We sent multiple random value-based queries and we calculated the average response times for the queries of the same interval length. Before executing each query, the database cache was flushed. As depicted in Figure 14, when data segments are indexed by the RI-tree method, an up to 40% improvement in response time can be achieved by the multi-model approximation algorithm, as compared to approximating the data set with Swing. On the other hand, as shown in Figure 15, when conventional database indexes are employed, multi-model approximation achieves an up to 24% improvement in response time. The achievable improvement in response time initally improves, then it gradually deteriorates, until it shows signs of convergence for larger data sets. This is explainable, because for small data sets, the use of indexes is ineffective for query answering, while their effectiveness increases as the tables grow. However, as tables grow, so does the answer for an interval query of a given length. Similar trends in the results are observed for the queries of the various selectivities considered. We performed similar experiments for point and time-based interval queries that verified that the fewer number of segments produced by the multi-model algorithm for approximating the data, as compared to single approximation models, results into more efficient data retrieval.

VIII. Conclusion

In this paper, we investigated the innovative concept of combining multiple models that approximate time series within a maximum error bound for achieving higher compression effectiveness than the individual models themselves. We proposed a greedy algorithm for finding an efficient model combination for high data compression and experimentally verified its effectiveness for all real and synthetic time series considered. Specifically, as found by the experiments up to 80% compression improvement can be achieved by our algorithm against individual approximation models. Moreover, we proposed an efficient database schema for storing and indexing the data segments produced by the models. When this storage schema is employed, our multi-model approximation algorithm is experimentally found to achieve 40% lower response time than that resulting by single-model approximation of the same data. As a future work, we intend to theoretically evaluate

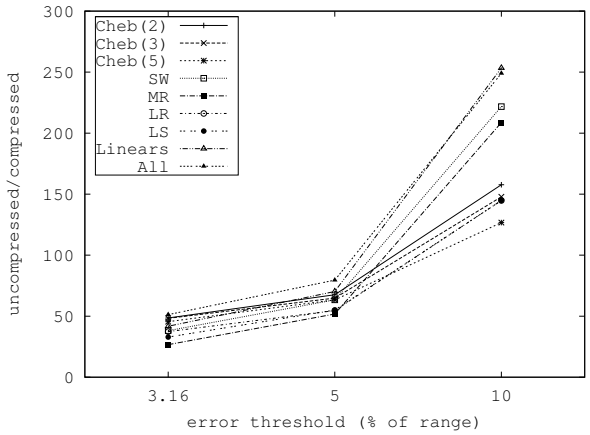

Fig. 13. The compression achieved by combining linear and polynomial models compared to combining only linear models (Ocean temperature dataset). For %10 error threshold combination of linear models outperforms the combination of all models.

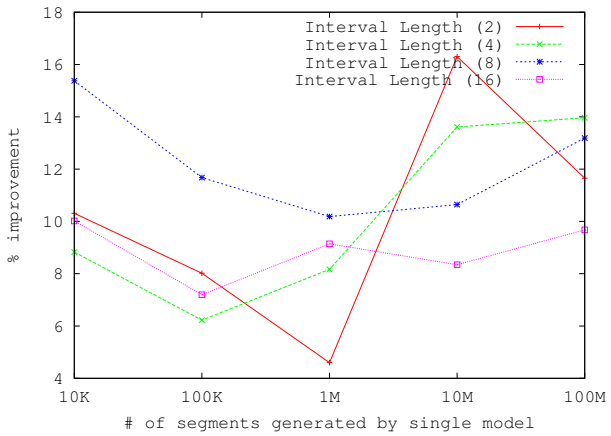

Fig. 14. Response time improvements using RI-tree indexing technique

the effectiveness of our algorithm as compared to the optimal combination of models for approximating the data segments of time series.

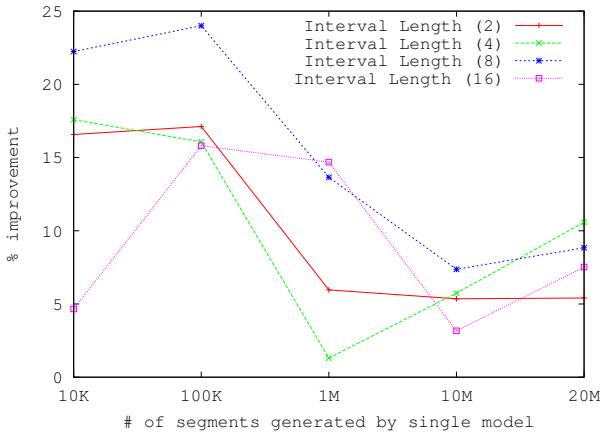

Fig. 15. Response time improvements only using database indexes

REFERENCES

[1] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref, and W. Zwaenepoel, "Online piece-wise linear approximation of numerical streams with precision guarantees," in *VLDB*, Lyon, France, August 2009.

[2] S. Gandhi, S. Nath, S. Suri, and J. Liu, "GAMPS: compressing multi sensor data by grouping and amplitude scaling," in *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, 2009, p. to appear.

[3] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Compressing historical information in sensor networks," in *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, 2004, pp. 527–538.

[4] R. T. Snodgrass, *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufman, 2000.

[5] S. Pattem, B. Krishnamachari, and R. Govindan, "The impact of spatial correlation on routing with compression in wireless sensor networks," in *Proc. of the Intl. Symposium on Information Processing in Sensor Networks*, April 2004, pp. 28–35.

[6] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *ICDE*, 2006, p. 48.

[7] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.

[8] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel, "Online amnesic approximation of streaming time series," in *ICDE '04 Proceedings of the 20th International Conference on Data Engineering*, 2004, pp. 339–349.

[9] I. Lazaridis and S. Mehrotra, "Capturing sensor-generated time series with quality guarantees," in *ICDE*, March 2003, pp. 429 – 440.

[10] A. Thiagarajan and S. Madden, "Querying continuous functions in a database system," in *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*. New York, NY, USA: ACM, 2008, pp. 791–804.

[11] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani, "An online algorithm for segmenting time series," in *ICDM '01 Proceedings of the 2001 IEEE International Conference on Data Mining*, 2001, pp. 289–296.

[12] S. Guha, "On the space—time of optimal, approximate and streaming algorithms for synopsis construction problems," *The VLDB Journal*, vol. 17, no. 6, pp. 1509–1535, 2008.

[13] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita, "Improved histograms for selectivity estimation of range predicates," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, 1996, pp. 294–305.

[14] L. Wang and A. Deshpande, "Predictive modeling-based data collection in wireless sensor networks." in *EWSN*, ser. Lecture Notes in Computer Science, R. Verdone, Ed., vol. 4913. Springer, 2008, pp. 34–51.

[15] J. Wu, Y. Zhou, K. Aberer, and K.-L. Tan, "Towards integrated and efficient scientific sensor data processing: a database approach," in *EDBT*, 2009, pp. 922–933.

[16] A. Deshpande and S. Madden, "Mauvedb: supporting model-based user views in database systems," in *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, 2006, pp. 73–84.

[17] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *ICDE*, 2008, pp. 1160–1169.

[18] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, "Probview: a flexible probabilistic database system," *ACM Trans. Database Syst.*, vol. 22, no. 3, pp. 419–469, 1997.

[19] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, 2003, pp. 551–562.

[20] S. Babu, M. Garofalakis, and R. Rastogi, "SPARTAN: A model-based semantic compression system for massive data tables," in *Proc. of the ACM SIGMOD Intl. Conference on Management of Data*, 2001, pp. 283–294.

[21] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963.