Efficient Model-Driven Query Processing Based on Data Regeneration

Alexandru Arion[§] Saket Sathe[§]

[§]Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
{name.surname}@epfl.ch

1. PROJECT DESCRIPTION

Nowadays, sensor data is generated in large amounts. Storing or transmitting all the sensor's measurements might not be the ideal choice because of the volume (and rate) at which it is generated. But we also cannot easily discard it, since every data might contain relevant information and be required while processing queries.

One approach to avoid storing and transmitting large amounts of data is to create a sufficiently accurate model over the data and then store or transmit that model. Later, at the time of query processing, the straight forward approach is to regenerate the required data from the model(s) and proceed with query processing, or answer the queries directly on the model(s). We ask the question of how computationally expensive are various query processing techniques, given the availability of the models only.

1.1 Project Aims

We are interested in comparing the computation costs of query processing for the following approaches:

1. direct query processing on original data (baseline performance);
2. data regeneration from the models, followed by query processing on the regenerated data;
3. direct query processing on models;
4. utilising GPUs for parallelization of operations wherever applicable. Specifically, using GPUs for model based data regeneration and direct query processing on models.

In this work we focus on evaluating these approaches for exact and probabilistic queries having aggregate functions [6]. For evaluation we plan to use multivariate time-series data originating from sensor networks. On reading this work, the interested reader will have an idea of the expected performance benefits of using various types of data models. These benefits could be very useful for deciding which models to employ in order to solve the space problem created by the large amount of data generated from sensors.

1.2 List of models

In this work we consider the following types of models:

- auto-regressive models
- moving average models
- interpolation polynomials

In our previous works, we have analyzed some of these models to compare the achievable compression factors. Since in a scenario of distributed storage of sensor readings, data transmission is a costly operation, we are rightfully looking for those models achieving highest compression ratio. In this project, on the other hand, we intend to add an orthogonal segment to this work, by evaluating efficient data regeneration and query processing techniques.

1.3 Measurement Metrics

We will measure the query execution time for some simple aggregate queries, exact or probabilistic:

```
SELECT [PREDICTED] avg(*)/max(*)
WHERE time < UPPER_BOUND
  AND time > LOWER_BOUND
```

2. RELATED WORK

Jampani *et al.* [9] have investigated Monte Carlo based strategies to generate data while query processing. They sample data from probability distribution(s) which the user has to specify in the query. In contrast to ours, their approach is not model-based and only supports basic probability distributions.

Deshpande *et al.* [4] have used models to perform data pre-selection in a wireless sensor network. They select which data from a wireless sensor network is worth bringing in at a consumer node. In their scenario query processing is performed directly on the pre-selected data, without considering model representation of it.

Graefe *et al.* [7] studies the benefits of data compression. They go beyond I/O performance, by adapting query processing algorithms to work on directly on compressed data. However, their compression scheme is rather different than ours: they employ an individual attribute level compression scheme, while we use models for column-wide data series.

In [5] Deshpande and Madden have proposed the use of model-based views. They construct database views estimating simple interpolation and regression models over raw data. Queries are then directly evaluated using these model-based views. In our approach we also estimate and store models but also consider evaluating queries on data regenerated from these models. We believe the latter approach is more suitable since, in general, it may not be feasible to directly evaluate queries over complex models. For example, certain non-Gaussian distributions (like, hyperbolic distributions) do not have closed formed expressions for their parameters which makes model-based query processing difficult.

3. TIME SERIES MODELS

Sensors continuously generate values for the parameters they monitor. Often the data at a given time could be modelled as a function of data received at previous times. For example, if a sensor is recording temperature values then the value recorded at time t would not be significantly different from the value at time $(t-1)$ since we know that temperature cannot change very rapidly. We can exploit this fact to build a time-series[1] model for the recorded values. Later, while query processing we use this model to regenerate the data for query response evaluation. For fitting a time-series model to data, we assume that a sensor records value r_t where t records the time and $t = 1, 2, \ldots, T$. Then a *moving average* model of order q for the time-series r_t is given as,

$$r_t = c_o + a_t - \theta_1 a_{t-1} - \cdots - \theta_q a_{t-q}, \tag{1}$$

where $a_t \sim N(0, \sigma_a^2)$, $\theta_1 \ldots \theta_q$ are the model coefficients, and c_0 is a constant. We denote a moving average model of order q as $MA(q)$. Moreover, in a moving average model a value at time t is an average of previous q values.

For estimating the model coefficients we could use either a *Maximum Likelihood Method (MLE)* or a *Conditional Least Square (CLS)* method [11]. Here, we use the CLS method due to it's relative simplicity and availability as standard routines in packages like MATLAB [2]. In the conditional least squares method we start from the $(q+1)^{th}$ observation. Now, the t^{th} sample is given as in (1), this forms a multiple linear regression problem whose coefficients are estimated using started objective function minimization techniques (for ex. Newton-Raphson Method). Let $\hat{\theta}_i$ denote the estimate of θ and $\hat{c}_0$ be the estimate of c_o then the fitted model is given as,

$$\hat{r}_t = \hat{c}_o + a_t - \hat{\theta}_1 a_{t-1} - \cdots - \hat{\theta}_q a_{t-q}. \tag{2}$$

Now the associated residual term $\hat{a}_t$ could be computed as $\hat{a}_t = r_t - \hat{r}_t$, which leads us to the estimation of $\hat{\sigma}_a^2$ as:

$$\hat{\sigma}_a^2 = \frac{\sum_{t=q+1}^{T} \hat{a}_t^2}{T - 2q - 1}. \tag{3}$$

Next, during the query processing stage we need to regenerate data using the $MA(q)$ model. This can be achieved as follows. Suppose we are at time index h and are interested in probabilistically regenerating the value at $r_{h+\ell}$ where $\ell \geq 1$, then the time index h is the *regeneration origin* and ℓ is the *regeneration horizon*. For a $MA(q)$ model a $\ell - step$ regeneration is given as:

$$r_{h+\ell} = c_o + a_h - \theta_1 a_{h+\ell-1} - \cdots - \theta_q a_{h+\ell-q}. \tag{4}$$

The $\ell - step$ ahead regeneration actually gives us the expected value of $r_{h+\ell}$, $E(r_{h+\ell}|F_h)$, where F_h is all the information available at the forecast origin h. Moreover, the moving average model regenerates *expected* or *most probable* values. This property makes them naturally robust against erroneous values and intermittent sensor failures.

The second type of time-series models we studied are the *auto-regressive models*. These models are similar to moving average models, but in auto-regressive models r_t is modelled as a linear combination of it's values at previous times. Thus a auto-regressive model of order p is given as,

$$r_t = \phi_0 + \phi_1 r_{t-1} + \cdots + \phi_p r_{t-p} + a_t, \tag{5}$$

where p is a non-negative integer, $\phi_0, \ldots, \phi_p$ are model coefficients, and a_t is as defined in (1). The model denotes that the conditional expectation of the value r_t given the past data is determined by the past p values $r_{t-i}, (i = 1, \ldots, p)$. We denote an auto-regressive model with order p as $AR(p)$. The estimation of $AR(p)$ model is done in a very similar way to the moving average model. We simply use (5) to evaluate $\hat{r}_t$ and $\hat{\sigma}_a^2$ in equations (2) and (3) respectively, which gives us an $AR(p)$ model.

Likewise, the auto-regressive model can be used for $\ell-step$ ahead regeneration similar to the moving average model. Moreover, a $\ell - step$ regeneration for an $AR(p)$ model is given as,

$$r_{h+\ell} = \phi_o + \phi_1 r_{h+\ell-1} + \cdots + \phi_p a_{h+\ell-p}. \tag{6}$$

Another important aspect of time-series models is the choice of the model parameters q and p. There are several criterion and heuristics proposed in the literature for making an optimal choice of model parameters for a given time-series. The most popular ones are based on the principles of entropy. These are known as the *Akaike Information Criteria (AIC)* and the *Bayesian Information Criteria (BIC)* [11]. For a given time-series these criterion output a score over all models obtained from increasing values of model parameters p and q for auto-regressive and moving models respectively. Then the most optimal model is the one which gives us the highest score for a given model order p or q. Admittedly, as the main focus of this project is not to estimate the optimal model order we choose a sufficiently large q for the moving average model. On the other hand, for auto-regressive models we compute the AIC for all the models upto a maximum model order p_{max} and then choose a p which gives the highest score. While performing experimental evaluation we set $p_{max} = 6$.

3.1 Evaluating Probabilistic Maximum Queries

Another important class of queries which the time-series could answer are the probabilistic min/max queries. This is possible due to the predictive regeneration capability of time-series models. Probabilistic maximum queries are queries which return a probability distribution over a set of possible value in a time-series. Suppose, for a time-series r_t we are at time h then a probabilistic maximum query could answer questions like, *"What is the most probable maximum value between time h and $h + \ell$?"*. For example, a probabilistic maximum query could be,

[1] In this project we collectively refer *moving average models* and *auto-regressive models* by *time-series* models.
[2] http://www.mathworks.com/

`SELECT prob_max(*) WHERE stream_id = SID AND horizon = TIME_HORIZON.`

If in the above query `TIME_HORIZON` is set to $h+\ell$ and `SID` is set to r_t, then the response is a probability distribution,

$$F_{max}(v) = \prod_{i=h}^{h+\ell} F_{r_i}(v), \tag{7}$$

where $F_{r_i}(v) = Prob(r_i < v)$, is the probability distribution of r_i. This assumes that $r_i, (i = h, h+1, \ldots, h+\ell)$ is conditionally independent given the past p values. From (7), the problem of answering such queries is narrowed down to estimating $F_{r_i}(\cdot)$ for $i = (h, \ldots, h+\ell)$. We estimate these distributions using a Monte-Carlo method.

We conduct τ trials by regenerating the values between h and $h+\ell$ according to model given by (5) (or (1)). For a given time i we denote the individual values obtained at various trials by $r_i^1, \ldots, r_i^\tau$. By using these τ values we obtain an estimate of $F_{r_i}(\cdot)$ as,

$$\hat{F}_{r_i}(v) = \frac{\sum_{s=1}^{\tau} \mathbb{I}_{\{r_i^s < v\}}(v)}{\tau}, \tag{8}$$

where $\mathbb{I}_A(x) = 1$ if $x \in A$ and $\mathbb{I}_A(x) = 0$ if $x \notin A$. Moreover, a parallel execution of the above evaluation is possible, since we can execute all the τ trials in parallel forming a matrix of size $\tau \times \ell$. Then, we can scan this matrix column-wise to obtain $\hat{F}_{r_i}(\cdot), (i = h, \ldots, h+\ell)$ which are used to evaluate $\hat{F}_{max}(\cdot)$. Actually, $\hat{F}_{max}(\cdot)$ is evaluated at discrete intervals $v = [v_1, \cdots, v_w]$ and a list of tuples $\langle v_1, \hat{F}_{max}(v_1)\rangle, \ldots, \langle v_s, \hat{F}_{max}(v_w)\rangle$ is returned as response to the query. Generally, in any Monte-carlo evaluation the number of trials which are required for obtaining considerable accuracy are very high since the estimation errors are inversely proportional to $\sqrt{\tau}$ [8].

3.2 Sampling Rate Adjustment

In both the time-series models we assumed that the values that we receive from the sensors are equally spaced in time. But, in practice this is not the case, which means that values arrive at non-equal arrival intervals. i.e. $t_{n+1} - t_n \neq t_{m+1} - t_m \forall n, m \in T$. Thus, to make the arrival interval equal we should re-sample at a fixed sampling interval. Now, the challenge is to choose a sampling interval such that a) information is not lost by choosing a large sampling interval and b) large number of samples are not generated by choosing a small sampling interval. Intuitively, we should choose a sampling interval such that the probability of having a value in this interval is maximized. To achieve this, we model the arrival times by a Poisson arrival process [10]. The Poisson process is characterized by a *rate* parameter (λ) which defines the expected number of arrivals in per unit of time. Thus, we choose $\frac{1}{\lambda}$ as the sampling interval for re-sampling our time-series data. Then we re-sample the original time-series by using linear interpolation at the sampling interval obtained from the method described above.

4. REGRESSION MODELS

The second major class of models we studied are *regression models*. For a given stream $s = \langle v_1, v_2, \cdots, v_k\rangle$, regression models describe how values $v_i \in s$ vary depending on the passage of time $t_i, i \in [1, k]$. The purpose of such a model is to estimate a value v'_{k+1} using the regression models when a new sensor reading v_{k+1} is streamed, so that a given accuracy constraint is obeyed: $|v_{k+1} - v'_{k+1}| < \epsilon$

Given a degree p, polynomial regressions find the best-fitting curve (or line when $p = 1$) that minimizes the total difference between the curve and each value v_i. They are defined as:

$$v_i = \alpha_0 + \alpha_1 \cdot t_i + \cdots + \alpha_p \cdot t_i^p + \epsilon_i, \qquad i = 1, ..., k \tag{9}$$

where α_j are regression coefficients and ϵ_i is the difference between the curve and v_i.

On the one hand, polynomials with higher degree can approximate the given stream with more sophisticated curves, rendering higher precision. On the other, higher-degree polynomials may involve more expensive data regeneration cost, longer construction time, and larger numbers of coefficients to be used for model update by our framework.

In our previous works, our experimental evaluation of the compression effectiveness of various degree polynomial regression, we found that best rates were obtained for linear ($n = 1$) models. Higher degree models required more space not only due to the additional coefficients that need to be stored per polynomial, but also due the frequency at which polynomial coefficients had to the updated, which was higher for higher degrees. Therefore we can conclude that the linear models will be the most effective also in terms of performance at query processing time, because of (1) the smaller size, which translates into less I/O, and (2) less computation required to regenerate data, since a low degree polynomial in cheaper to compute than a high degree polynomial. Therefore we have focused evaluation of query processing performance on linear models.

4.1 Query evaluation with data regeneration

In the context of our problem, we have the polynomial models as input, i.e. their coefficients and the time intervals for which a given polynomial fits properly the data stream. Computing the data values from the model means evaluating the polynomial at the required time instances. Given the independence of evaluations, this type of model fits well to parallelization using a programmable GPU.

4.2 Query evaluation on model

The query classes we have considered (aggregate queries) can be answered directly on the model, without any data regeneration required. Average over a collection of polynomials can be computed by the following approach: (1) computing the local average on each polynomial; (2) giving it the right weight according to the time interval the polynomial is meant to represent the actual data stream compared to the other polynomials; (3) computing the weighted average of the local averages. Computing the local average for a polynomial $P(X)$ with X between a and b can be done using the formula: $AVG(P, a, b) = \int_a^b P(X)dX/(b-a)$

5. INDEXED RAW DATA

For comparison purposes, we have considered evaluating how less computationally expensive is to answer queries on raw data if the raw data is indexed on the variables present in the `WHERE` clause of the query, in our case, *time*. We have created an index which permits direct random access to the disk location which contains the value of interest, therefore reducing I/O for low selectivity queries.

6. EVALUATION FRAMEWORK

A relevant piece of our work was to design the software framework that permits us to incrementally develop and incorporate various parameters of the project, such as

- modeling techniques of the data
- decompression methods (i.e: GPU-aided, index-optimized, direct query answer on models)
- query types
- data sources

The above concepts form the skeleton of our framework, and each of them represents a distinct replaceable component, that eases future development. The current framework is developed using Java [3].

Additional separate features were added to help in measuring performance metrics.

7. EXPERIMENTAL RESULTS

Overview.
Experimental evaluation provides answers to the following questions:

- How does query processing efficiency change when using models as input, rather than using the raw data, without employing any other optimization technique (i.e: we used data regeneration)?
- How much do additional optimization techniques improve performance?

To answer these question we have defined the following testing procedure:

1. Select a time-series corresponding to a stream that is available to us from a real sensor-network deployment.
2. To ensure fairness in comparing raw data and models query processing efficiency, we decided to store them in similar formats: as files on disk. Therefore we dump raw data in one file and corresponding model representation in another file.
3. Flush the disk's and filesystem's caches but after this do a dry-run of our java code, to ensure we measure only query processing time, and not JavaVM page-faults when instantiating another object in the VM.
4. Run our code for various selectivity levels and query processing approaches of the following query:

```
SELECT avg(*) WHERE time < UPPER_BOUND AND
time > LOWER_BOUND
```

5. Measure time spent in reading the required data from disk and time spent in building the answer to the query.

We run the test on 4 time-series: soil-moisture and air temperature at 2 different physical locations in the Grand St. Bernard area in Switzerland. Measurements were taken for the duration of 6 months, but in our tests we reduced the used time-period to 1 week. The configuration of CPU and GPU used for experimental evaluation is given in Table (1).

[3]http://java.sun.com/

7.1 Polynomial Models

In the following paragraphs we evaluate and compare several approaches for processing exact aggregate queries using polynomial models.

Regeneration Based.
We have generated linear-models for the 4 time-series, using an accuracy bound (ϵ) of 1 unit (in a range of values of approx. 60 units).

We have aggregated our results in figure 1 (a). We make several comments on the results to give insight into which are the factors influencing performance of query evaluation based on models.

First, different data-sets show different improvement factors. This is explainable by the differences in model sizes and raw data sizes. The smaller a model is compared to the raw data, the lower the I/O cost, and therefore smaller time cost for processing query. Figure 2 shows how speed-up is correlated to compression ratio, for the smallest selectivity (0.05), which ensures that the I/O cost is the primordial factor of influence (no index is used at this stage of our experiments).

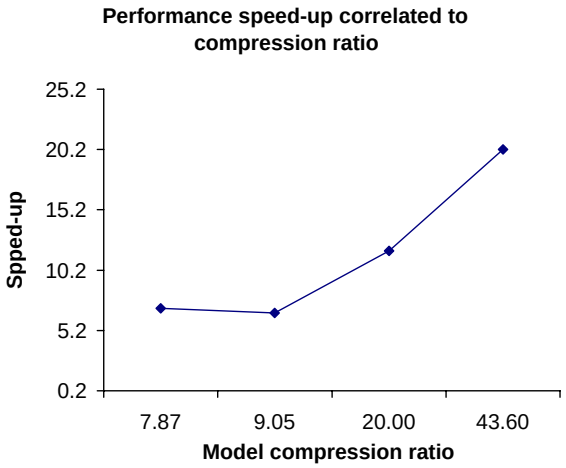

Figure 2: Speed-up correlation to compression ratio for linear regression models.

Second, the speed-up decreases as we increase selectivity. This is explained by the fact that increased selectivity means an increase in the *average* computation time for both the model-based processing and the raw data processing and also an additional computation cost for the model-based processing, as more data has to be generated (no optimization is applied yet: data that is needed is regenerated, even tough, as we will see further, the can compute the average without regenerating).

Direct Query Answer.
We considered worth investigating if avoiding data regeneration would bring significant benefits to model-based query processing. For the computationally simple polynomial models we found out that I/O cost is responsible for almost all cost (99%) of a query processing, therefore optimizing the CPU time (avoid regeneration) would have very little impact. However we compared the efficiency of non-optimized (w. data regeneration) and optimized (w/o data regeneration) of model-based query processing assuming no I/O.

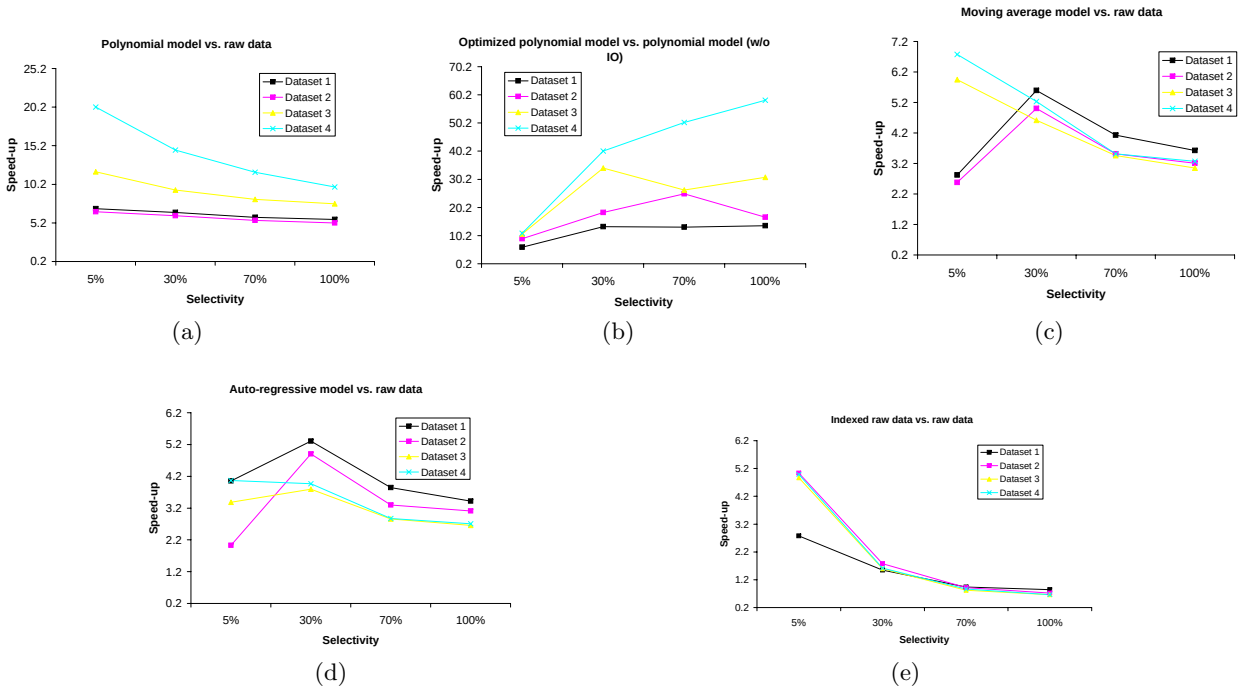

Figure 1: Comparing speedups of model based and raw data query processing: (a) polynomial model vs. raw data; (b) optimized polynomial model; (c) moving average model; (d) auto-regressive model; (e) indexed raw data.

	CPU	GPU
Model	Intel Xeon	NVIDIA Tesla C870
Memory	16 GB	1.5 GB
Number of cores	8	128
Clock rate	2.66 GHz	1.35 GHz
Bandwidth	–	2 GB/s

Table 1: CPU and GPU configuration used for experimental evaluation.

Results are summarized in 1 (b). It's worth pointing out the reason for the differences in speed-up among data sets: each dataset has a corresponding set of models. The amount of data values regenerated in the non-optimized approach is the same for all datasets, but the number of local average computations in the optimized approach is proportional to the number of models used for each dataset.

GPU parallelization.
The last tackled performance improvement approach was making use of the parallelizable characteristic of the optimized polynomial technique. Computing the local averages for each polynomial part of the model for a given time-series can be done in parallel for all polynomials. By using the highly parallel programmable GPUs available on the market, we measured the performance speed-up (or lack of), the displayed the results in 3.

This graphs shows a beautiful non-linear system, caused by the following components:

1. CPU, where for large number of models, compiler optimizations like loop unrolling increase cache performance and therefore cost varies sub-linearly with the size of the input.

2. Memory transfers between main memory and GPU memory, were latency predominates up to a certain point the transfer time, and is gradually replaced by bandwidth, as data transfer size increases.

3. GPU, because of the large number of concurrent threads, the full parallel processing capacity is reached only for very large number of models.

But the answer we were looking for is that total GPU time is smaller than total CPU time only between 2 values for the number of models. As can be seen in the graph, the predominant reason for which GPU is not faster than CPU is that the code to be executed in parallel is fast, and therefore memory transfer dominates the total GPU computation time.

Indexed Raw Data.
An additional interesting comparison is between the optimized raw data access (index-based) and the non-optimized model-based access. Results are present in 1(e) and show that the speed-up obtained by using index is smaller than that obtained by using models.

7.2 Time Series Models

In the following paragraphs we evaluate and compare regeneration based approaches for answering exact and probabilistic queries using time-series models.

Exact Queries.
For each time-series containing W values we estimate a $MA(q)$ model on a sliding window of H. Then we slide the window

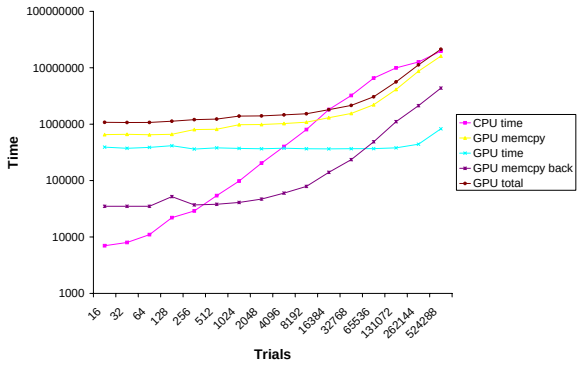

Figure 3: Query processing costs for CPU and various GPU operations for exact queries on polynomial models.

by the number of values for which we intend to use the model. Thus, for each time-series we estimate $\lfloor \frac{x}{W} \rfloor$ models, if we intend to use the model for generating x values. We estimate the models using MATLAB and store them in a CSV (comma separated values) file along with the time for which the query processor should be using the model. In our experiments, we set $H = 200$ and $x = 100$ for all the time-series. The value H is selected such that we have sufficient data for model estimation. The same procedure is used for estimating an $AR(p)$ model of suitable model order as explained in Section (3).

Next, we use the estimated models to compare the performance of evaluating the same queries on raw data without using an index. Figure (1).c and Figure (1).d show the speed-up obtained (w.r.t. processing queries on raw data without index) from using moving average model and auto-regressive model respectively for data regeneration. The overall speed-up is less as compared to the linear regression models since the cost of data regeneration is high for the time-series models (for example, at each regeneration step we have to sample from a normal distribution). But, on the other hand, time-series models are more robust towards erroneous readings as they probabilistically track the true value. As compared to the indexed raw stream the time-series models exhibit higher speedup, which could make them more useful especially when selectivity is high. Most importantly, as discussed in Section (3.1), time-series models can be effectively used for answering probabilistic queries, this could be difficult to achieve by using other models.

Probabilistic Queries.
In the following paragraphs we discuss and compare the evaluation of probabilistic maximum queries on CPU and GPU (graphics processing unit). The GPU implementation possess several challenges, we discuss some of them below:

Parallel random number generation: For conducting τ trials we need parallel random numbers generators for the term a_t (see equations (1) and (5)) which is present in both the time-series models. If the same random number generator is used for all the τ trials then the generated random numbers could be correlated with high probability. Thus we implement a 32-bit parallel *Multiply-With-Carry (MWC)* random number generator [3]. We choose the parameters and seeds such that the random numbers generated are uncorrelated [2].

Coalesced Accesses: While we simulated the $\tau \times \ell$ matrix we always ensured that the accesses to global memory are coalesced [1]. Since for NVIDIA C870 coalesced global memory accesses are orders of magnitude faster than non-coalesced ones, but they can only occur only when certain requirements are satisfied by a group of threads (for details see [1]).

Furthermore, we compare performance of CPU based query evaluation to GPU based query evaluation for various values of trials τ. For comparing performance we take an $AR(p)$ model and use it to evaluate probabilistic maximum queries for a fixed value of ℓ and various values of τ. Recall, in practice τ is large since the estimation errors are inversely proportional to $\sqrt{\tau}$. Thus we can expect the values of τ to be much larger than ℓ, making $\tau \times \ell$ a tall matrix. Figure (4) shows the speedup obtained by using a GPU based implementation. Clearly, as the number of trials increase the speedup obtained by the GPU based implementation increases thus making it more suitable for Monte-Carlo based probabilistic query evaluation. Furthermore, we expect the same speedup for probabilistic *minimum* queries since they incur the same computation as the probabilistic maximum queries.

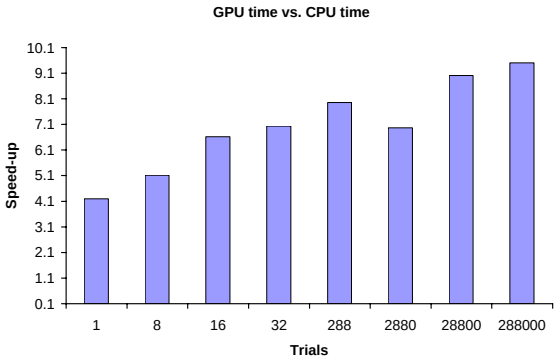

Figure 4: Comparison of performance between CPU and GPU for probabilistic maximum queries.

8. FUTURE WORK

Considering the range of queries that a data management system supports, analyzing the applicability of our proposed approaches and evaluating the computational cost for other query classes is necessary. This would lead us to providing a metrics which helps in choosing a particular modeling technique.

9. CONCLUSION

We answered the question of how computationally expensive are various query processing techniques on compressed data, and the benefits that can be gained by employing various optimized approaches. Our evaluation showed that the smaller size of the models, compared to the raw data, gives substantial performance benefits due to the lower I/O (5x-20x). With all data in memory, direct model based query for computationally simple models (polynomial) shows good improvements compared to the data regeneration approach (10x-50x). The use of a programmable GPU for parallel data regeneration provides large improvements (4x-10x) for computationally intensive queries (probabilistic min/max queries).

This is mainly due to the memory transfer time from main memory to GPU memory being less significant than actual model computation time.

10. ACKNOWLEDGEMENTS

Our thanks go to Prof. Anastasia Ailamaki for the feedback and suggestions provided throughout the course of this work.

11. REFERENCES

[1] NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide, Version 2.0. http://www.nvidia.com/object/cuda_develop.html.

[2] Steven Gratton's webpage on Parallel Random Number Generation. http://www.ast.cam.ac.uk/~stg20/cuda/random/index.html.

[3] R. Couture and P. L'ecuyer. Distribution properties of multiply-with-carry random number generators. *Mathematics of Computation*, 66(218):591–608, 1997.

[4] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 588–599. VLDB Endowment, 2004.

[5] A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 73–84. ACM New York, NY, USA, 2006.

[6] R. Elmasri and S. Navathe. *Fundamentals of database systems*. Addison-Wesley Reading, Mass, 2000.

[7] G. Graefe and L. Shapiro. Data compression and database performance. In *Applied Computing, 1991.,[Proceedings of the 1991] Symposium on*, pages 22–27, 1991.

[8] J. Hammersley and D. Handscomb. *Monte carlo methods*. Methuen Young books, 1964.

[9] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 687–700, New York, NY, USA, 2008. ACM.

[10] S. Ross. *Introduction to Probability Models*. Academic press, 2007.

[11] R. Shumway and D. Stoffer. *Time Series Analysis and its Applications*. Springer, New York, 2006.