A Skip-list Approach for Efficiently Processing Forecasting Queries

Tingjian Ge, Stan Zdonik
Department of Computer Science
Brown University
{tige, sbz}@cs.brown.edu

ABSTRACT
Time series data is common in many settings including scientific and financial applications. In these applications, the amount of data is often very large. We seek to support prediction queries over time series data. Prediction relies on model building which can be too expensive to be practical if it is based on a large number of data points. We propose to use statistical tests of hypotheses to choose a proper subset of data points to use for a given prediction query interval. This involves two steps: choosing a proper history length and choosing the number of data points to use within this history. Further, we use an I/O conscious skip list data structure to provide samples of the original data set. Based on the statistics collected for a query workload, which we model as a probability mass function (PMF) over query intervals, we devise a randomized algorithm that selects a set of pre-built models (PM's) to construct, subject to some maintenance cost constraint when there are updates. Given this set of PM's, we discuss interesting query processing strategies for not only point queries, but also range, aggregation, and JOIN queries. We conduct a comprehensive empirical study on real world datasets to verify the effectiveness of our approaches and algorithms.

1. INTRODUCTION

1.1 Motivation and Our Solution

There has been recent interest in forecasting queries [10]. Scientific, financial, and business applications rely on time series data [30, 28]. Decision making often requires forecasting over time series data at different time scales. The following three example areas illustrate (1) short-, (2) medium-, and (3) long-term forecasting requirements respectively.

(1) *Scheduling*: Forecasts of the level of demand for various products are an essential input to near-term scheduling of production, transportation, and personnel.

(2) *Acquiring resources*: Forecasting is needed to determine future resource requirements in order to plan for acquisition lead times that could span several months.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-305-1/08/08

(3) *Determining resource requirements*: Forecasts of financial, human, and technological requirements are helpful for determining what resources an organization will need in the long-term.

In these applications, the amount of data is often very large. Consider the time series of trades and quotes (called ticks). Stock quotes arrive every second. Financial analysts want to predict stock prices minutes ahead, hours ahead, days ahead, months ahead, or sometimes years ahead. Our goal is to use a moderate amount of data for any *prediction interval* (a term which we use to mean how far into the future we predict), so that we can explore a larger portion of the huge space of possible models to find a model that can serve as an accurate predictor. Note that we use the term "*forecast*" and "*prediction*" interchangeably in this paper.

A simple example of a forecasting query is the following:

SELECT * FROM ticks
WHERE symbol = "IBM" and time = NOW + 1 day

Clearly, excessive granularity of data is unnecessary and inefficient or even impractical for a given prediction interval. For example, to predict the stock price of some company one year from now, it is wise to use a *history length* of a certain number of years (say, 20 years). Too short a history may give a partial picture of the evolution of the stock data, thus making the prediction result inaccurate [15]. On the other hand, too long a history length may not offer more useful information for the prediction, and sometimes may even complicate and disturb the model building [31], thereby, also reducing accuracy.

Forecasting requires a *model* of the time series that is a function of the form $v = M(t)$ where t is time and v is the value at time t. For the types of models discussed in this paper, selection of model M is like curve fitting. A history length of 20 years with one tick per-second has $20*365*24*3600 = 630{,}720{,}000$ values! A typical model selection and building process is expensive, and using this large number of data points is impractical. In fact, even for predicting 15 days from now (using, say, a 12-month history), the required history length would still be prohibitively large with over 30 million values.

Thus, one might not want to use such a fine granularity as one value per second for predicting something one year or even 10 days in the future. There have also been studies in the statistics and forecasting literature on the *minimum* number of data points required for forecasting (e.g., see [14, 16]). For the prediction of a specified interval, we choose a subsequence embedded within the original time series as a "new" time series of a different "time granularity". In summary,

- We may use different "absolute history lengths" for different forecast intervals *f*.
- Given a history length *h(f)*, we determine the number of data points *n* to use for model building.

We use a *skip list* data structure [24] to provide fast data access for different levels of granularity. In addition to supporting *prediction*, a skip list also supports *searching* (i.e., indexing). Each level of the skip list has a set of models (i.e., prediction functions) associated with it. We can also build models at the leaf level of a skip list to *interpolate* missing data values in the past. Note that the searching and interpolation aspects are straightforward and the focus of this paper is on prediction of various future intervals using data at different levels of the skip list.

The original skip list data structure is only meant to be in memory. To be scalable for large data sets, it needs to be stored on disk. We adopt it in our context and discuss its organization on the disk.

Different levels of a skip list have different data densities. For a given query interval *f*, as we discussed earlier, we can determine a proper history length *h(f)* to use and the number of subsequence data points *n* to use within *h(f)* for model building. Thus, *n/h(f)* gives a *data density* which we use to select a level of the skip list that has the closest density.

Throughout the paper, we use *multiple regression* models to illustrate our ideas for a couple of reasons:

- Practice shows that it gives good results and is widely used [15]. There have been more than 35 surveys among forecasting users [29, 17] since 1970. Regression is the method that users have the highest level of satisfaction with among all time series forecasting methods.
- It does not require data points to be equidistant, which makes it more flexible and more widely applicable to database applications in which data is not limited to time series data.

We note that it is straightforward to extend the ideas in this paper to other forecasting models.

If characteristics of the workload are known, we can pre-build a set of models for prediction queries using our skip list technique. If the workload is unknown, we can build the models on the fly. We must also consider the maintenance costs for updating the pre-built models as new data comes in. It is worth noting that on-line performance will be improved using our skip-lists when we must either dynamically build models or frequently maintain (rebuild) the models under update.

We present a randomized algorithm called *ChoosePMSet* to select a set of models to pre-build subject to a maintenance cost constraint. This constraint is based on query interval workload information described as a PMF (Probability Mass Function). A prediction query is hence answered by picking the "closest" pre-built model (PM) to use. We measure how well the set of PM's "serves" the workload by computing the expected *model distance* (which we define in Section 4) of a prediction query. The PM for prediction queries are analogous to materialized views (MV) for traditional queries. The key difference is that an MV materializes the data tuples while a PM only "materializes" the parameters of a model (e.g., coefficients of a polynomial), which is highly compact.

Using PM's for query processing is more straightforward for point queries than for more complex query types. We discuss query processing techniques using PM's for interesting query types, namely, range queries, aggregations, and join queries. We avoid materializing future data points for efficiency. Finally, we perform a comprehensive empirical study on two real world datasets to verify the algorithms and approaches set forth in the paper.

To sum up, the contributions of this paper are:

- A proposal for using the skip list data structure to build models that simultaneously provide search, interpolation, and prediction (SIP) capabilities.
- A proposal of using PM (Pre-built Models) to efficiently process prediction queries. A randomized algorithm to effectively select a set of PM's to build, subject to a maintenance cost constraint. We analyze how well the set of PM's produced by the algorithm serves a given query load.
- Query processing techniques using PM's.
- A comprehensive empirical study using real world data sets.

1.2 Related Work

In the context of *online and streaming* applications, there has been previous work (e.g., [7] and [21]) that addresses a similar problem to ours, namely, query processing when there is a large amount of historical data. Bulut and Singh (in [7]) develop a technique using Discrete Wavelet Transform that summarizes a dynamic stream incrementally at multiple resolutions. Palpanas et al. (in [21]) introduce the notion of general amnesic functions which describe the precision loss for queries on different periods in the past.

The work in [7, 21] concerns *online streaming* in which large amounts of historical data must be discarded, while our work is aimed at *stored data*. Often, fine-granularity historical data is needed for queries. Also, in the case of stock ticks or medical databases, there is often a regulatory requirement to store all the data. These days, large amounts of data are being generated by measurement infrastructures that continuously monitor a variety of things like military object positions or environmental properties. In these examples, the data volume is huge. Searching, for existing values, interpolating missing values, and predicting future values are all important. The skip lists in our solution can be used for searching and interpolation in addition to prediction, making them more general than [7, 21].

Furthermore, [7, 21] addresses general queries on the past (point, range and "inner product" queries) while our work aims specifically at forecasting queries of various types: point, range, aggregations, and join. For forecasting queries, our skip list approach is simpler and more efficient in that (1) the database engine does not need to pay any computation overhead associated with maintaining and *transforming* data summaries; (2) the approach in [7, 21] has to discard some recent data points to build a model that uses data points (almost) equidistant in time in order to ensure that the *least square error metric* for optimization is fair for all time periods in the chosen history length.

In fact, forecasting using data of higher sample frequency is a known problem in the literature [2]. In particular, the study in [2] shows that the improvement of forecasting results using higher sampling frequency can be quite dramatic. The skip list approach provides a platform to explore data of different densities.

The skip list data structure was invented in 1990 by Pugh [24]. Its elegance and simplicity have drawn a lot of attention. Munro et al. [19] proposed a deterministic version to guarantee logarithmic costs. Aspnes and Shah [4] proposed skip graphs, which are a distributed structure based on skip lists, and provide the functionality of a balanced tree in a distributed system for fault tolerance. Abraham et al. proposed an improved version, so-called "skip B-trees", that combines the advantages of skip graphs with features of B-trees. There is also a project called "skipDB" [33] which is a database implemented with a skip list instead of a B-tree. It is claimed to be transactional, portable, fast and small.

Time series is one of the primary special data types required within scientific databases [30]. There has been a lot of work, especially in data mining, on similarity and pattern matching in time series. To list but a few, work along these lines includes [12, 23, 32, 22, and 13]. Time series forecasting has been a major focus for research in other fields. In particular, valuable tools for forecasting and time series processing appear in statistics and signal processing. [8] is a recent and comprehensive review of this research over the past 25 years. [6] and [11] present additional work in this area.

In the context of databases, Yi et al. [31] developed a fast method to analyze co-evolving time sequences jointly to allow estimation or forecasting of missing/future values, quantitative data mining, and outlier detection. Tulone and Madden [27] presented a method for approximating the values of sensors in a wireless sensor network based on time series forecasting. Also in the context of sensor networks, Deshpande and Madden [9] developed view abstraction for the underlying interpolation and prediction models to support declarative queries. More recently, Duan and Babu [10] developed algorithms that can compose prediction operators into a good plan for a given query and dataset.

Our work differs from earlier work in important ways. We focus on the data management aspects, specifically, the scalability issue for predictive query processing when the time series data set is large. This is crucial for query performance as well as prediction accuracy since typically model building is expensive. We target the issue of choosing the right subset of data to answer prediction queries on a given future interval. We also discuss interesting query processing strategies for handling complex query types, whereas in [10], for example, only point queries are supported, but not other query types such as range query, aggregation, and join. Last but not least, our skip list approach also simultaneously provides search and interpolation capabilities.

The remainder of this paper is organized as follows. In Section 2, we give the background knowledge of this work. Section 3 shows the building blocks of how we use skip lists to provide SIP functionalities. We discuss how to use *statistical tests of hypotheses* to determine a proper history length and the number of data points within this history to use for building a model for a given query interval. We show how to select a set of models to pre-build in Section 4, query processing techniques in Section 5, and an extension to multiple independent variables in Section 6. Section 7 presents the experimental results. Finally, we conclude the paper in Section 8.

2. BACKGROUND

2.1 Skip Lists

A skip list [24] is a randomized counterpart of a balanced tree structure, such as the B+ tree. As shown in Figure 1, the skip list structure supports levels, each of which is a linked list of sorted keys. It is quite simple: a key at level *i* also appears at level *i+1* with probability *p*. Thus, logically, insert and delete are rather simple: insert into leaf level first, and then toss a coin such that with probability *p* we also put it in the next higher level. The procedure stops when it fails to appear in some level. Deletion of a key *k* simply removes *k* from all levels. Probabilistically, it is equivalent to a tree of fan-out *1/p*. The balance of the equivalent tree is automatically maintained by the magic of probability.

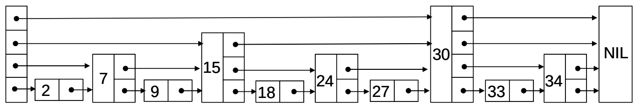

Figure 1. Illustrating the skip list data structure.

We search for an element *e* by traversing pointers until we either find *e* or we overshoot the node containing *e*. When we overshoot *e* at the current level, the search moves down to the next level starting at the node with the largest found key less than *e*. For example, in Figure 1, suppose we want to search for the node with key 27. We start at the head of the top level, find the pointer to 30 which overshoots 27. Then we go down to the next level, follow a pointer to 15, then the pointer to 30 which again overshoots. Next, we begin at node 15 and go to the lower level, follow the pointer to 24, then the pointer to 30 which overshoots. Finally, starting at node 24, we descend to the bottom level and find 27.

2.2 Forecasting Models

As we discussed in Section 1, forecasting queries are useful for applications in many domains. One can extend SQL with additional constructs like a keyword NOW that can be used in a query such as "*SELECT price FROM ibm_ticks WHERE time = now + 10 days*" which would produce a prediction of IBM's stock price ten days from now.

To answer forecasting queries, people build mathematical *models* on the *existing data*, trying to find the trends that reflect the underlying governing factors. Using such a model one would expect to make a reasonable forecast of a value in the future. Forecasting has been closely related to *time series* data (a sequence of observations x_t , each one being recorded at a specific time *t*), although it can be applied to variables other than time. Time series forecasting is a well-studied area in statistics. There are a rich set of models that statisticians have developed for this purpose. We refer readers to some excellent introductory books, e.g., [6, 11, 15]. Below, we briefly introduce *multiple regression* models as a forecasting method since we use it in this work, for the reasons we discussed in Section 1.

Multiple regression (a.k.a. linear regression), a time-honored technique going back to Pearson's use in 1908, is a regression method that models the relationship between a dependent variable *Y*, independent variables X_i, $i = 1, \ldots, p$, and a random term ε. The model can be written as $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$. Note that Y is called a "dependent" variable (i.e., dependent on X_i's) and X_i's are collectively called "independent" variables, even though they may not be independent among themselves. It is often erroneously thought that the reason the technique is also called "linear regression" is that the graph is a straight line or that Y is a linear function of the X variables. But if the model is, say, $Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$, it is still a multiple (linear) regression, that is, linear in x and x^2 respectively, even though the graph on x by itself is not a straight line.

Thus, power terms can be added as independent variables to explore curvilinear effects and cross-product terms can be added as independent variables to explore interaction effects. In general, the function can be a polynomial of any degree. This is often called *polynomial fit* which is a specific type of multiple regression. Usually, *least squares estimates* are used to obtain the coefficients β_i. For time series data, the function would be a polynomial on the *time* variable. We refer readers to any statistics textbook (e.g., [16]) for details. As mentioned in Section 1, surveys show that multiple regression, as a prediction tool, is widely used and the most satisfactory in practice [15].

2.3 Statistical Tests of Hypotheses

In statistics, there are two general methods available for making inferences about *parameters* based on some population data. We can estimate the parameter values with *confidence intervals* or we can *make decisions* about the truth or falsity of some statement about the parameters by *testing hypotheses* (i.e., making decisions). A statistical test of hypothesis consists of four elements [16]:

(1) **Null hypothesis**, H_0, about one or more population parameters,
(2) **Alternative hypothesis**, H_a, that we will accept if we decide to reject H_0,
(3) **Test statistic**, computed from sample data, and
(4) **Rejection region**, indicating the values of the test statistic that will imply rejection of the null hypothesis.

To see a simple example, suppose an investigator for the Environmental Protection Agency (EPA) wants to determine whether the mean level μ of a type of pollutant released into the air by a chemical company exceeds a limit in order to determine whether the company is violating the law. Testing by hypotheses is a method analogous to proof by contradiction. The theory the EPA wants to support (alternative hypothesis) is that, say, $\mu > 10$. The theory contrary to that (called null hypothesis) is that μ is at most equal to 10. The EPA may use a sample of $n = 30$ daily pollution readings. If the sample mean $\bar{y}$ (test statistic) is much larger than 10 (rejection region), the EPA would tend to reject the null hypothesis and conclude that $\mu > 10$. Again, we refer readers to standard statistics textbooks (e.g., [16]) for details.

3. ELEMENTS OF OUR APPROACH

3.1 I/O Conscious Skip Lists

We adopt the skip list data structure in our context, and make it I/O conscious. As stated earlier, time series databases can be too large to fit in memory. For example, 20 years of per second stock quotes have about *630M* data points and reach gigabytes. Thus, for scalability, we need to consider the efficiency of query processing when storing a skip list on disk.

The original skip list structure requires a large number of pointers, which is detrimental for I/O performance. In model building for prediction queries, we use a contiguous sequence of data at some level of the skip list. A search operation, as described in Section 2.1, also accesses a contiguous sequence of data at each level. Thus, we replicate key values at each level and store them compactly and contiguously in disk pages, instead of using pointers (one for each level) on only one copy of keys as in the original skip list. Time series data associated with the keys are stored together on pages. For example, in our stock example, *time* is the key and the *(time, stock price)* pair is stored in the skip list. Clearly, for the search to proceed, we need to store, for each key value, a pointer to its copy in the level below. Figure 2 illustrates this.

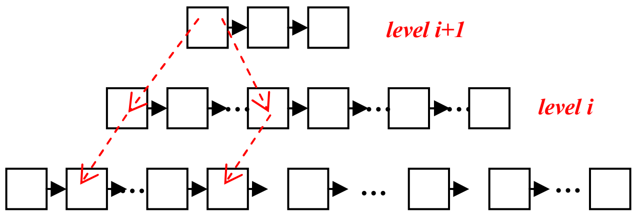

Figure 2. Illustrating the I/O conscious skip list structure. Each node in a linked list represents a disk page of keys and associated values. Dashed arrows (only four of them are drawn) represent pointers from a key to its appearance in the lower level, used for search.

We can handle overflow and underflow of pages when there are updates using “open” and “closed” pages, in the same manner as in [1, 3]. We omit the details due to space limitations. The basic idea is to maintain an invariant that requires every page to be filled within a percentage range. Time series data updates are mostly “appends” [15], which makes merging and splitting of pages rare. For append only data sets, we simply keep adding pages at each level, and possibly removing pages at the other end of a level of skip list when the oldest data is no longer relevant.

Note that unlike a B+ tree, whose fan-out is fixed by the database page size, the parameter p of a skip list is flexible, which we need for different sample data point densities. Furthermore, key values at each level of a skip list are chained together, unlike a B+ tree. We use these features of a skip list to efficiently retrieve samples with some needed probability from a level of the skip list to build forecasting models.

3.2 Prediction Models

As we mentioned in Section 1, searching and interpolation with a skip list are straightforward. For searching, a skip-list only helps predicates over the history based on its sort key (e.g., *time=10*). If the desired data points are missing, we have models for interpolation. Searching is the basic functionality provided by a skip list; interpolation occurs only at the base level of the skip list and is a well-studied problem. We refer readers to [20, 9] for some of this work in databases. Therefore, from now on, we only discuss prediction using the skip list approach.

As we discussed in Section 1, for a given prediction interval, we pick a level of the skip list to build a model. We shall present the method of how to pick a level and how many data points in that level to use in Section 3.3 and 3.4. Given that, since we always use data up to the most recent for answering prediction queries, we use a suffix of some level in building a model. Thus, a given level of a skip list can have 0 or more associated models, each of which is built with a different suffix sequence.

3.3 Determining a Proper History Length

In this section, we first study the issue of how to determine a proper history length *h(f)* to use for a given forecast interval *f*. The basic idea is that we use a small number of most recent data points as the target *training set*, and "go back in time", starting from the earliest point in the training set, for an interval *f* (denoting that point in time as T_{-f}). We then determine a proper history length *h'* going further back (i.e., from T_{-f} to $T_{-f-h'}$) from which we can predict the target training set data well. We determine *h'* using *statistical tests of hypotheses*. Figure 3 illustrates this. The algorithm is shown in Figure 4.

For multiple regressions, $F = s_{i-1}^2 / s_i^2$ has an *F distribution* with n_{i-1}-k_{i-1}-1 *numerator degrees of freedom* and n_i-k_i-1 *denominator degrees of freedom* [16]. Thus, each iteration of the loop conducts a statistical test of hypotheses with H_0 being "use h_{i-1}" and H_a being "use h_i". If H_a is true, then F is big. The rejection region is $F > F_\alpha$. The stopping condition (line 8) is to stop the loop at a point in the final downward slope of the F distribution. Intuitively, the algorithm iteratively increases the history length and runs statistical tests of hypotheses, until it determines that any further increase in history length "is not worth it".

An implicit assumption here, of course, is that for a given forecast interval *f*, if we "went back" in time for a period of *f*, and could use some duration of history data points relative to that time to "predict" the "present" time data points (thus the forecast interval is also *f*), then we can use this data to predict accurately the "real interval *f* into the future" (illustrated in Figure 3).

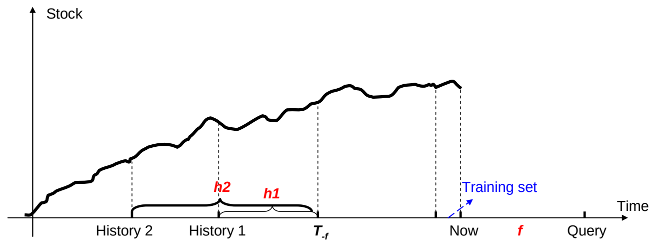

Figure 3. Illustrating the determination of a proper history length.

The following table summarizes the parameters of the algorithm:

c_t	The number of data points in the training set *T*.
c_0	A parameter that determines the initial history length ($c_0 f$).
c_1	Multiplicative increment of history length ($h_i = c_1 h_{i-1}$).
F_α	Threshold that determines the rejection region of the testing of hypotheses: $F > F_\alpha$.

```
Input: A forecast interval f of a query.
Output: A proper history length h(f) to use for answering the query.
(1) Set the most recent $c_t$ data points as the target training set T, whose values we use other data points to "forecast" (to be able to compare the "forecast" values with the actual ones).
(2) Let the smallest time value in T be $T_0$. Let $T_{-f} = T_0 - f$.
(3) Set $h_0 = c_0 f$. Use standard techniques [14] to build an optimal multiple regression model using data points in $[T_{-f} - h_0, T_{-f}]$ and compute its mean square error $s_0^2 = SSE_0/(n_0-k_0-1)$, where $SSE_0$ is the sum of squared error, $n_0$ is the number of data points used, and $k_0$ is the number of parameters in the model. Let $i = 0$.
(4) Do
(5)     $i = i + 1$; $h_i = c_1 h_{i-1}$.
(6)     Use standard techniques to build an optimal multiple regression model using data points in $[T_{-f} - h_i, T_{-f}]$ and compute its mean square error $s_i^2 = SSE_i/(n_i-k_i-1)$.
(7)     $F = s_{i-1}^2 / s_i^2$.
(8) While $F > F_\alpha$.
(9) Output $h_{i-1} + f + |T|$, where $|T|$ is the time length of T.
```

Figure 4. An algorithm that uses statistical tests of hypotheses to determine a proper history length.

3.4 Determining the Number of Data Points

Note that in the above algorithm, we use all available data points within a trial history length to build a multiple regression model. We have shown in Section 1 that this is often too expensive for building and maintaining models, and the excessive granularity is actually unnecessary and wasted. Thus, a natural approach is to sample and use a subset of all the available data points. Studies in the statistics and forecasting literature are concerned with the *minimum number of data point requirement* for forecasting (e.g., see [14, 16]), which is just a lower bound and using it may still give bad prediction results.

Therefore, the basic problem is: given *f* and *h(f)* (determined by the above algorithm), how do we determine the number of random points to use within *h(f)*? The idea is similar to the previously presented algorithm to determine the proper history length; thus we omit the details. Roughly, we iteratively increase the number of random points used in *h(f)* for building a trial model, and again we use statistical testing of hypotheses to determine a good choice of the number, within a reasonable computational cost constraint.

4. SELECTION OF MODEL SET TO BUILD AND MAINTAIN

4.1 Basic Working Model

We organize the time series in question into a skip list. The skip list has a parameter ***p***, which is the probability that an element in a lower level is also present in the next higher level. We choose a set of models to pre-build at various levels of the skip list (i.e., Pre-built Models, or PM's). Query processing picks one or more closest PM's to use, or could even build a model on the fly. The interesting aspects between PM's and skip list levels are:

- A PM uses a suffix sequence of the data points of some level.
- A level can have 0 or more PM's.

We also maintain the set of PM's we have chosen to pre-build when new data comes in or when updates happen. More specifically, a model is rebuilt whenever *both* $\boldsymbol{\theta}$ (a threshold parameter) new data points have entered the model *and* the model is used by some query. Thus, it is a *lazy maintenance* strategy. There is a constraint on the total model rebuilding cost as described below. A model update involves using the same number of the most recent data points at the level of that model in the skip-list to rebuild the regression model. In addition, after a sufficient number of new data points enter the model, we choose the history length and the number of data points again.

4.2 Quantifying Model Maintenance Cost

We next quantify the maintenance cost of a set of models. We assume a set of models in a skip list that we have chosen to build and maintain. New tuples arrive at some rate.

Theorem 1. *We organize time series data into a skip list with parameters p, θ and the lazy maintenance strategy as described in Section 4.1. New tuples come in at a rate of r (tuples/sec), and we consider the expected incoming rate for upper levels of the skip list. Let the set of models be M. For a model* $m \in M$ *, let l(m) be the skip list level at which the model is located and q(m) be the reference rate (times/sec) of the model by queries. Let* C_R *denote the canonical rebuilding cost of a model. Then with the tuple incoming rate, the maintenance cost rate of M is*

$$C_R \cdot \sum_{m \in M} \frac{1}{\max\left(\frac{1}{q(m)}, \frac{\theta}{r \cdot p^{l(m)}}\right)}.$$

Proof: For a model *m* located at level *l(m)* of the skip list, the arrival rate of new tuples for that level is $r \cdot p^{l(m)}$. The *lazy maintenance* strategy implies that a model is rebuilt either when every θ new data points come in, or when the model is used by some query, whichever happens later. Thus, a model is rebuilt every $\max\left(\frac{1}{q(m)}, \frac{\theta}{r \cdot p^{l(m)}}\right)$ seconds. Then it is clear that the overall maintenance cost rate is

$$C_R \cdot \sum_{m \in M} \frac{1}{\max\left(\frac{1}{q(m)}, \frac{\theta}{r \cdot p^{l(m)}}\right)}. \quad \square$$

Note that the optimal history length and the number of data points to use for a given prediction interval length may change as time progresses. We consider this as part of the model rebuilding (i.e., an ingredient of C_R in Theorem 1). A system can choose these parameters again after a certain number of new data points enter the model.

4.3 Choosing a Set of Models to Build

We are only concerned with the set of forecast intervals of a query workload. Thus, we model the query workload as a discrete PMF *w* on forecast intervals $f_i = \frac{iF}{n} \; (1 \le i \le n)$, with their associated probabilities $p_i \; (1 \le i \le n)$, respectively, where *F* is the maximal forecast interval.

The optimization problem is that given a query workload, subject to a constraint on maximal maintenance cost, we want to find a set of intervals for which we build models so that the *expected model distance* (defined in Section 4.4) for a random query in the workload is minimized. Note that different models use different levels of the skip list and can have different maintenance cost (Theorem 1). This problem is similar in spirit to the *knapsack problem* (but with the extra complication that the *value* of an item is correlated with what items are being selected). Thus, an efficient optimal algorithm is unknown. Because randomized algorithms are known for their simplicity and efficiency [18], we devise such an algorithm, to provide a practical solution and to make theoretical analysis easier (Section 4.4). In fact, because of its efficiency, one can repeat the algorithm several times to choose the result with the smallest expected model distance. Figure 5 shows the algorithm (called *ChoosePMSet*).

```
Input: a query workload w as a discrete PMF; a constraint on maximal model maintenance cost rate $C_M$.
Output: A set of forecast intervals for which we build models.
(1) Let $M = \Phi$.
(2) Repeat
(3)     Obtain a random sample of forecast interval $f$ from query workload PMF $w$, using a standard method to sample from a discrete distribution.
(4)     $M = M \cup \{f\}$.
(5)     From $f$, determine the proper history length $h$ and the number of data points $n$ to use within the history length using algorithms in Section 4. From $h$ and $n$, we get the density of the data points. Thus, a model will be built using the skip list level that has the closest density.
(6)     Incrementally compute the maintenance cost rate $C$ of the set $M$ using Theorem 1.
(7) Until $C > C_M$ or $M$ contains all intervals.
(8) If $(C > C_M)$ then $M = M - \{f\}$.
(9) Output $M$.
```

Figure 5. The *ChoosePMSet* algorithm that selects a set of PM's to build, subject to some maintenance cost rate constraint.

The algorithm repeatedly samples a new forecast interval *f* from the workload PMF *w* using established weighted sampling methods from a discrete PMF. It continues this process until the maintenance cost rate of the models exceeds the constraint.

Analogous to the database design problem for materialized views, this kind of pre-built structure often requires knowledge of the statistics of future requests. The statistics are collected through profiling at the database server, etc. Although PM's can be robust against certain changes of the workload, a rebuild is unavoidable when dramatic changes occur. As an input of *ChoosePMSet*, distribution *w* can reflect how much knowledge of the workload is assumed. Less knowledge implies a "flatter" distribution while more knowledge renders a more specific distribution.

4.4 Analysis of the *ChoosePMSet* Algorithm

We next analyze "how well" the workload PMF *w* is satisfied after running the algorithm *ChoosePMSet* to produce a set of models to build and maintain within the cost budget. To be precise, we need the following definition.

Definition 1. *Let the output M of ChoosePMSet have m forecast interval points out of a total of n points* $f_i = \frac{iF}{n}\ (1 \le i \le n)$ *where i is called the* index *of a point. Then for an arbitrary query point* $f_i = \frac{iF}{n}\ (1 \le i \le n)$ *define its* model distance *as the index distance between* f_i *and the closest point in M.* □

For example, for query point f_{95}, if the closest point in *M* is f_{99}, then the model distance of f_{95} is $99 - 95 = 4$.

Theorem 2. *Let m and n be as described in Definition 1. Then the expected model distance of a query point in workload w is* $\sum_{i=1}^{n}\sum_{d=1}^{n-1} p_i (1 - \sum_{j=i-d}^{i+d} p_j)^m$

Proof. For a query point with *index i*, define random variable D_i as its *model distance*. Then the probability that none of the *m* independent samples falls in a radius *d* of the query point *i* is,

$$\Pr[D_i \ge d] = \left(1 - \sum_{j=i-d}^{i+d} p_j\right)^m \qquad (1)$$

As D_i is a discrete random variable with *non-negative* values, we have (intuitively, for d from 1 upwards, cumulatively, $\Pr[D_i \ge d]$ is the probability that we add 1 to the expectation [18]),

$$E(D_i) = \sum_{d=1}^{\infty} \Pr[D_i \ge d] = \sum_{d=1}^{n-1} \Pr[D_i \ge d] \qquad (2)$$

From (1) and (2), we have $E(D_i) = \sum_{d=1}^{n-1}\left(1 - \sum_{j=i-d}^{i+d} p_j\right)^m$

Define random variable *D* as a random query point (in *w*)'s *model distance*. Thus,

$$E(D) = E(\sum_{i=1}^{n} p_i D_i) = \sum_{i=1}^{n} p_i E(D_i) = \sum_{i=1}^{n}\sum_{d=1}^{n-1} p_i (1 - \sum_{j=i-d}^{i+d} p_j)^m$$

where the second equality follows from the linearity of expectation. □

As we shall show in Section 7.3 (Figure 11-b), we can write a simple program to compute the expected model distance for a specific instance of the problem.

5. QUERY PROCESSING

In this section, we discuss query processing techniques with a PM set. In general, for a query on future time series data, we pick the closet pre-built model to use. This is clearly straightforward for point queries. We discuss interesting query types, namely, range query, aggregations, and joins.

5.1 Range Queries and Aggregations

We discuss aggregation queries (in particular, SUM/AVG and MIN/MAX) with a range predicate, as that would include the treatment of both range queries and aggregations.

5.1.1 SUM/AVG with a Range Predicate

Let us start with an example query:

Q1: SELECT AVG(price) FROM ibm_ticks WHERE time ≥ now + 10 days AND time ≤ now + 30 days

A trivial way to evaluate such a query is to "materialize" all future data points in the range of the predicate, and then compute the aggregate in the brute-force way. However, it turns out that there are much more efficient ways. For that, we first demonstrate an axiom called the *monotonicity assumption*.

Monotonicity Assumption. *When the forecast interval f increases, we can assume that the optimal history length h(f) also increases or stays the same, and the data point density of the model used either decreases or stays the same.* □

Intuitively, the monotonicity assumption makes sense because to predict a longer interval, one wants to use a longer history length, with a sparser granularity of the data points. Since the data point density drops when the skip-list level increases, we have the following corollary.

Corollary 1. *For a forecast interval f, let m(f) denote the pre-built model we use to answer f, and accordingly, l(m(f)) denotes the skip-list level of the model. Then, when f increases, l(m(f)) either also increases or stays the same.* □

As the prediction interval increases, the level (in a skip list) of the model used must either go up or stay the same (in which case the number of data points used does not drop). Thus, there is a total order of all the PM's, consistent with the order of query intervals.

From the corollary, we can see that a range query is answered by *a set of contiguous models* (in terms of their skip-list levels), each answering a sub-range of the predicate. We shall verify the validity of the monotonicity assumption empirically in Section 7.3 and 7.4 (Figure 12 and 13).

Theorem 3. The *result of a basic SUM query with a range predicate for a future time interval* $[t_0, t_k]$ *as in Q1 can be computed as*

$$\sum_{t=t_0}^{t_1} f_1(t) + \sum_{t=t_1+1}^{t_2} f_2(t) + \ldots + \sum_{t=t_{k-1}+1}^{t_k} f_k(t)$$

where $f_i(t) = \sum_{j=0}^{d_i}(c_{ij} \cdot t^j),\ 1 \le i \le k$ *are a set of contiguous polynomial regression models in the skip list.* □

As the sum of powers of integers is a well-studied problem in mathematics [5, 25], we can compute the *SUM/AVG* with time complexity *O(kd)*, where *k* is the number of models spanned by the range predicate, and *d* is the maximal degree of any of those models. Since typically both *k* and *d* are small constants, we achieve constant time complexity. This is in contrast to the naive method of materializing every future data points, which requires a linear processing cost.

Example 1. *Suppose a range predicate like the one in Q1 spans three models and the sum can be represented by the following:*

$$\sum_{t=10}^{15}(3t^2-7t+10)+\sum_{t=16}^{22}(-0.1t^3+11t^2-t+9)+\sum_{t=23}^{30}(8t^2-15t+2)$$

It is known that

$$s_1(n)=\sum_{i=1}^{n} i=\frac{n(n+1)}{2}, \qquad s_2(n)=\sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6},$$

$$s_3(n)=\sum_{i=1}^{n} i^3=\frac{n^2(n+1)^2}{4}$$

Thus, the sum can be rewritten as

$$3(s_2(15)-s_2(9))-7(s_1(15)-s_1(9))+60-0.1(s_3(22)-s_3(15))$$
$$+11(s_2(22)-s_2(15))-(s_1(22)-s_1(15))+63+8(s_2(30)-s_2(22))$$
$$-15(s_1(30)-s_1(22))+16$$

and we obtain the result for sum. □

5.1.2 *MIN/MAX with a Range Predicate*

We now look at the MIN/MAX aggregations in a range of a future time interval. Consider this example query:

Q2: SELECT MAX(price), MIN(price) FROM ibm_ticks WHERE time ≥ now + 10 days AND time ≤ now + 30 days

To answer a MAX aggregation over a future time range, consider the simple case that the time range is covered by only one model. Let f be the polynomial function of the multi-regression model. For a continuous function, to get the maximum [26], we want to find a time value t, such that

$$\frac{df}{dt}=0 \qquad (1)$$

$$\frac{d^2f}{dt^2}<0 \qquad (2)$$

Most functional relationships in nature seem to be smooth (except for random errors) – that is, they are not subject to irregular reversals in direction. So the degree of the polynomial is generally low [16], most often 1 to 3, rarely greater than 3. In fact, a high degree often indicates over-fitting and is not a good model. Skip-lists reduce the data points and avoid over-fitting. Thus, in practice, computing roots for (1) and (2) is easy and there are not many solutions.

However, we actually have a set of *discrete* time values and a peak value we find from solving (1) and (2) may not fall in the set. In that case, we call the two closest time values in the set *discrete peaks*. For example, suppose a model spans the range *[10, 20]*, but one of the solutions from (1) and (2) is $t = 16.3$, then $t = 16$ and $t = 17$ are the "discrete peaks". We also note that the time range of the query can span multiple models. The following theorem determines the result for a MAX or MIN query.

Theorem 4. *Let* $M = \{f_0(\cdot), f_1(\cdot),..., f_{k-1}(\cdot)\}$ *be the set of k contiguous regression models spanned by the range predicate of a MAX query. Let the range of the predicate be* $[t_0+1, t_k]$ *and each model* $f_i(\cdot)$ *covers the sub-range of* $[t_i+1, t_{i+1}]$. *We call* t_0+1, t_1, t_1+1, t_2, $t_2+1,..., t_k$ *the* borders *of M. Then for the MAX query, we only need to examine the discrete peaks (if any) of each of the k models and the borders of M. A MIN query can be answered analogously by changing the inequality in (2) to ">".*

Proof. Suppose that the MAX value were not a discrete peak or a border of *M*. Let the time of the MAX value be t and let it be in model f. It must be true that both $t – 1$ and $t + 1$ are also in f, since t is not a border. Because f is a continuous function and t is not a discrete peak, it must be true that either $f(t – 1) \geq f(t)$ or $f(t + 1) \geq f(t)$. Thus, we could use either $t – 1$ or $t + 1$ as the MAX. The same argument repeats until we reach either a border or a discrete peak. □

5.2 JOIN Queries

We now look at JOIN queries with JOIN predicates on values in a future time range. Consider this query:

Q3: SELECT ibm.day, ibm.stock, sun.day, sun.stock FROM ibm, sun WHERE ibm.day BETWEEN (now, now+30days) AND sun.day BETWEEN (now, now+30days) AND ibm.stock > sun.stock

A naive way to answer a JOIN query of a future time range is to generate all future data points in the range for both relations, and then determine a JOIN strategy using a classical optimizer. However, a much more efficient way is to do a "model JOIN".

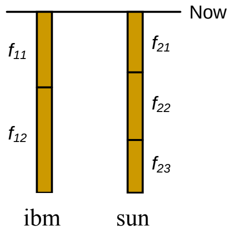

Figure 6. Illustrating the "model JOIN".

As shown in Figure 6, for each model of one *relation* in the query range (f_{11} and f_{12} of ibm), we solve an inequality or equality (depending on the JOIN predicate). In this example, we solve $f_{11}(t) > v$, i.e., say, $3t^2 – 6t + 5 – v > 0$. Likewise, we solve $f_{12}(t) > v$, etc. Thus, for each value in the query range of the second relation, we use the solution of the inequalities/equalities (i.e., $f_{11}(t) > v$ and $f_{12}(t) > v$, etc.) to get the matching tuples in the first relation. Clearly, this is just a linear cost overall, and is much more efficient than materializing the data points.

6. MULTIPLE INDEPENDENT VARIABLES

Instead of "time", there can be multiple independent variables. For example, in a military application, the x coordinate and y coordinate of a tank's position can be two variables, and the tank's speed v is the dependent variable. Thus, we can build a multi-regression model for v, dependent on x and y. In terms of the organization of the skip list, the general idea is that we only create one set of samples for all the independent variables, but link the samples differently for each variable.

In more detail, let the k independent variables be $x_1, x_2, ..., x_k$. Let the dependent variable be v. Thus, a multi-regression model is $v = f(x_1, ..., x_k)$, where f is a polynomial function. A data point can be represented as $(x_1, ..., x_k, v)$. Each level of the skip-list data structure is created by sampling as before. As in the single dimension case, there is a parameter p for the skip list which is the probability that a data point in a lower level also appears in the upper level.

Moreover, we create a separate link-list at each level of the data structure for each of the *k* independent variables. Thus effectively, there is a separate skip-list structure for each variable, since in general each variable has a different sort order. Yet these skip lists share the same physical nodes; each of them just follows a different set of pointers. Note that typically, there are not many independent variables, and we cluster the data points on the first independent variable (hence no pointers needed for it), which is the most likely to be used in a search.

The advantage of having an ordered list for each independent variable is that we can search on any of them. In case of searching on multiple independent variables (i.e., multiple predicates), we can perform parallel probes of the skip list according to each variable. Finally, at the base level, we do an "AND" to get the intersection, much like index AND'ing in traditional B+ tree indexes. It is worthwhile noting that searching on a non-clustered independent variable involves more I/O cost, but it is still O(log*n*) in expectation.

It is easy to see that with straightforward extensions, the algorithms we discussed for the single variable case work for multiple variables too.

7. EMPIRICAL STUDY

In this section, we study empirically the following questions:

1. How effective is the skip list approach? How does it compare with building models directly on original data in terms of speed and prediction accuracy?
2. How effective is the set of PM's (pre-built models) selected by the algorithm *ChoosePMSet* for answering prediction queries? How does it compare with building models on-the-fly in terms of prediction accuracy and speed?
3. What is the real tradeoff between prediction accuracy and different numbers of PM's (for different PM maintenance cost constraints)? Does this match the model distance of queries as calculated by Theorem 2?
4. Is the monotonicity assumption, which we use to process complex query types, valid for real data?
5. How does the number of PM's (chosen by *ChoosePMSet*) affect the result of complex queries such as aggregation with range predicates?

7.1 Setup and Datasets

We implemented the skip list approach, the algorithms and query processing techniques presented in this paper. The experiments were conducted on a 1.6GHz AMD Turion 64 machine with 1GB physical memory and a TOSHIBA MK8040GSX disk. The implementation is in Java. We performed the experiments on two sets of stock price data (from Commodity Systems, Inc.).

1. IBM's stock price history data from January 3rd, 1966 to October 10th, 2007. This per second tick dataset is over 1 GB.
2. McDonald's stock price history data from January 2nd, 1970 to October 10th, 2007. This dataset also has almost 1 GB of tick data.

The data is already normalized through the adjusted price. In order to verify the accuracy of predictions of different length of future time intervals, we go back one year in history and pretend it is now October 10th, 2006. We use data up to this date to build models and predict stock ticks at different "future" time intervals relative to October 10th 2006, for up to one year. Then we can use the actual stock prices from October 10th 2006 to the same day in 2007 to verify the accuracy of predictions using various methods.

7.2 Effectiveness of the Skip List Approach

In the first experiment, we examine the effectiveness of the skip list approach for answering prediction queries.

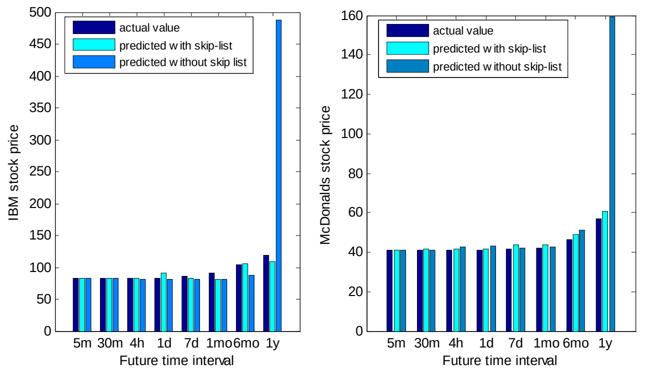

Figure 7. Comparison of actual value and prediction results with and without the skip list approach for different prediction intervals.

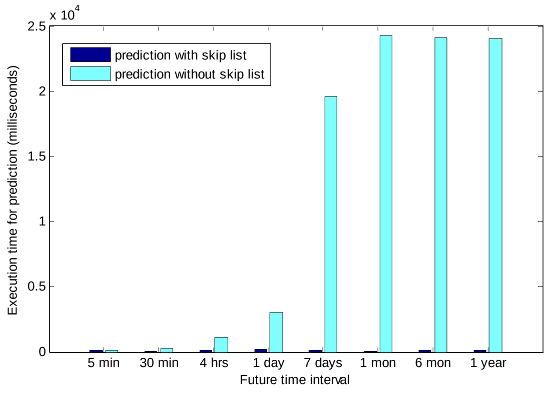

Figure 8. Comparison of query execution time with and without the skip list approach.

We issue queries of the form "SELECT stock_price FROM IBM_ticks WHERE time = NOW + ?". Figure 7 shows the prediction accuracy with and without the skip list approach for seven prediction intervals. The figure shows the results for both datasets. Without using the skip list (the third bar), we directly build models on the original dataset and apply a limit (*500,000*) on the maximum number of data points that can be used, which we will explain next. Figure 8 compares the query execution time with and without the skip list approach.

From Figure 7, we can see that as prediction interval increases, the quality of prediction without the skip list approach drops rapidly. The reason is as follows. We showed in Section 1 that the "proper" history length increases with the prediction interval. Because a skip-list supports efficient retrieval of samples at different granularities, model building and, thus, model maintenance as well, only reads the necessary data as opposed to reading all the raw data within the same history length. Furthermore, accessing a level in the skip-list is a sequential scan requiring no additional seeks. Figure 8 shows that even for a short prediction interval of 4 hours, the query execution time without the skip list is already a few times longer since we are building models on the fly. We apply a limit (*500,000*) on the maximum number of data points used because (1) when beyond this limit, the model building takes so much memory and CPU that it runs too slowly on our test machine; and (2) at this limit it is already more than 300 times slower than using a skip list. Figure 8 indicates that when the prediction interval is one month or longer we already reach this limit. Figure 7 also shows when the query interval is *one year*, the history length available (subject to the limit on maximum number of data points) without using skip lists is too short to make a meaningful prediction.

We use $p = 0.05$ for the skip lists (p is described in Section 2.1) in the experiments of this section. For two p values (say, between 0.01 and 0.5) p_1 and p_2, the ratio of their space costs is approximately $1+ p_1 + p_1^2 + p_1^3$ over $1+ p_2 + p_2^2 + p_2^3$. This is because if the base level raw data takes space S, then the second level takes space approximately $S*p$, the third level takes $S*p^2$, and so on. High order (above 3) terms can be ignored from the ratio. In our experiments, as we vary p between 0.01 and 0.5, the query performance did not change much. This is because the server always accesses the level of the skip list closest to the desired density to build a model. Prediction accuracy, however, can be affected because the distance between the desired data density and the actual data density used in the skip-list is affected by p. This is analogous to how *expected model distance* affects accuracy. In Section 7.3 we study the correlation between model distance and accuracy.

7.3 Effectiveness of the PM's

In the second experiment, we examine the effectiveness of answering prediction queries using a set of PM's chosen by the *ChoosePMSet* algorithm subject to different levels of maintenance cost constraints. Typically, we assume that through profiling at the database server, for example, we can collect some statistics on the query workload, a PMF over a set of query intervals. In the experiment, we pick 27 intervals in the one-year window, ranging from 5 minutes to 1 year. We test with an arbitrary PMF, shown in Figure 9. Again we look at accuracy and speed.

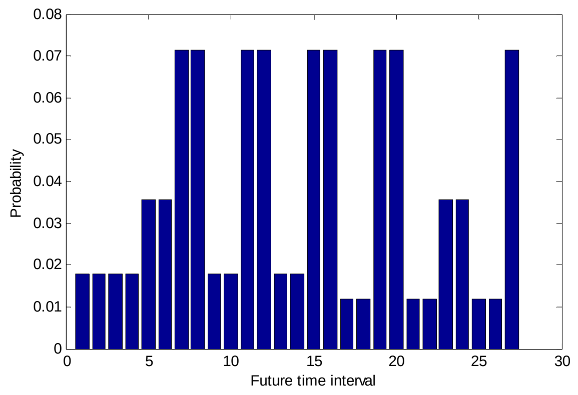

Figure 9. Probability mass function (PMF) of future time intervals as the workload.

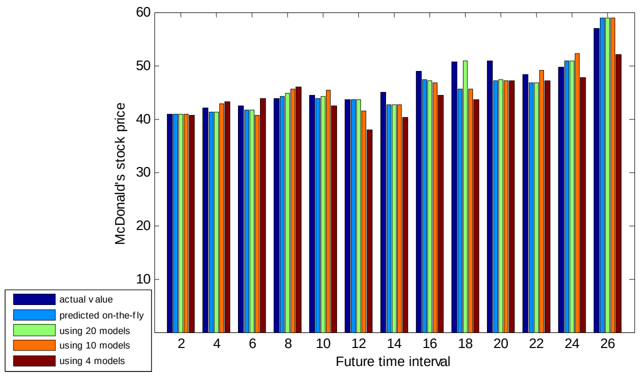

Figure 10. Prediction accuracy using different number of PM's.

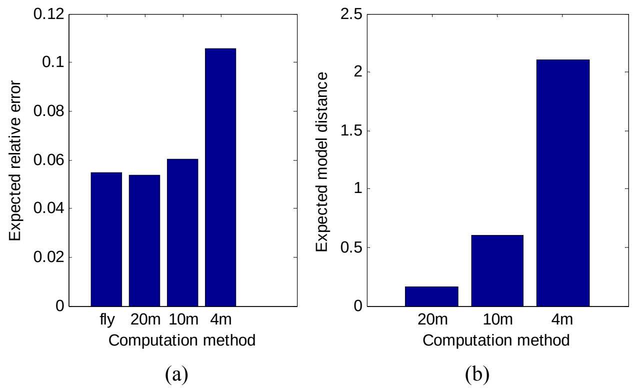

Figure 11. Expected prediction error (a) and expected model distance metric computed using Theorem 2 (b) of using different number of PM's.

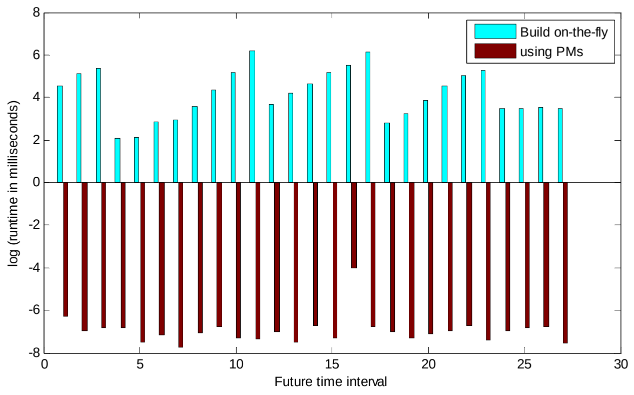

Figure 12. Query execution time comparison between building models on-the-fly and using PM's.

Figure 10, 11 and 12 show the results for the McDonald's dataset (Due to space constraints, we omit the figures for the IBM dataset in which we see similar trends). Each group of Figure 10 has five bars. They are the *actual value*, *predicted on-the-fly*, *using the first 20 models* selected by the *ChoosePMSet* algorithm, *using the first 10 models*, and using *the first 4 models*, respectively. For clarity, we only show the even number intervals (the other half shows similar information). Using 20 models is about as good as building models on the fly. Using 10 models is in fact also very close to this "best-we-can-do" result, and using 4 models is sometimes quite inaccurate compared to others.

Figure 11(a) summarizes the expected relative error of Figure 10 (but all 27 intervals) according to the workload PMF. The first bar is for building models on-the-fly to answer a query, and the other three bars are for answering with 20 PM's, 10 PM's and 4 PM's, respectively. We can see that the error of using 20 PM's is about the same as building models on the fly. Using 10 PM's is nearly as good, but using 4 PM's has significantly more error. Figure 11(b) simply plots the result from our theoretical analysis in Theorem 2 of the expected model distance of an incoming query. The model distance with 10 PM's is close to 20 PM's, while using 4 PM's has significantly bigger model distance. This is consistent with the result of Figure 11(a) on prediction errors.

Figure 12 compares the execution time of answering queries (for different intervals) by building models on the fly versus using PM's. Since the running time of using PM's, regardless of how many of them, is about the same, we only show one of them. Here we observe that the query processing time using PM's is negligible compared to building models on-the-fly. For both bars to be visible, we use log base 2 for the *y* axis. From the running time of building models on-the-fly, we can also observe that there are five groups (intervals 1 to 3, 4 to 11, 12 to 17, 18 to 23, and 24 to 27), within each of which the on-the-fly running time monotonically goes up (or stays about the same). Each of the five groups corresponds to the usage of a different level of the skip list, and within each level, as the query interval goes up, the history length, hence the number of data points used also goes up, which causes model building time to go up. This partially verifies the monotonicity assumption (Section 5.1). We further verify the skip list level part of the monotonicity assumption in Section 7.4.

7.4 Monotonicity Assumption and Query Processing

In the third experiment, we verify the monotonicity assumption we made in Section 5 for query processing, and examine the prediction accuracy, as well as the stability (variance) of result, of an aggregation query using 10 PM's and 4 PM's.

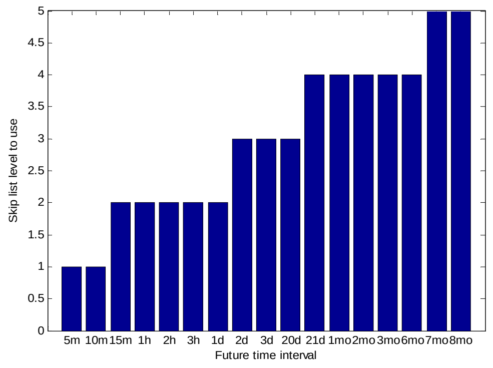

Figure 13. Monotonicity of skip list level used for different query intervals.

Figure 13 shows that using our statistical testing of hypotheses algorithms for determining a proper history length and number of data points to use, we determine a skip list level to use which is monotonically increasing as interval goes up. We can narrow down the exact transition intervals that make a jump of the skip list level.

We next look at the processing result of an average query "SELECT AVG(stock_price) FROM IBM_ticks WHERE time BETWEEN NOW AND NOW + ?" using 10 and 4 PM's. Figure 14 shows the result for the IBM dataset. Due to space constraints, we omit the figure for the McDonald's dataset, as it leads us to the same conclusions. We have three runs using 10 PM's and three runs using 4 PM's. In each run, we start from *ChoosePMSet*, which is a randomized algorithm. Thus the result of query processing is also a random variable. The first bar in each group of Figure 14 shows the actual average value, and the next three bars are the results of three runs of using 10 PM's, and the last three bars are those of using 4PM's. We can see that using 10 PM's predicts the aggregation result pretty well, and the results of the three runs are close to each other, which indicates that the query processing result from 10 models is quite stable. On the other hand, using 4PM's, the result is about the same as 10 PM's in expectation. The 4 PM case, though, has a much larger variance, and hence the prediction result can be far off.

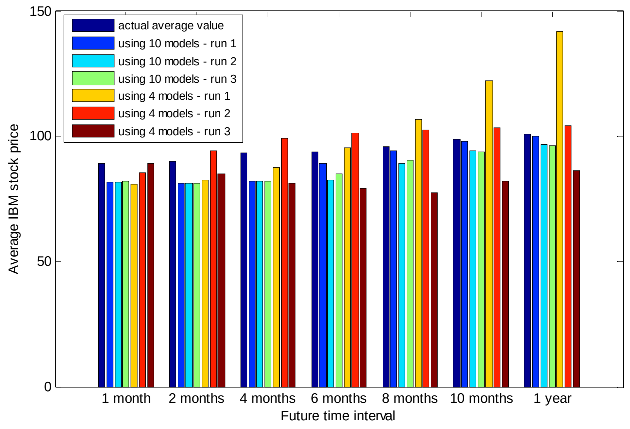

Figure 14. Comparison of the prediction accuracy of average stock prices in different future time intervals under different number of PM's.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we address the scalability issue on processing prediction queries on large time series data sets, which are often seen in financial and scientific databases. We propose statistical tests of hypotheses to determine a proper subset of data points to use for a given query interval. We adopt the skip list data structure, make it I/O conscious, and use it as samples for our query purpose, in addition to the search capability that a skip list already provides. We further present an algorithm *ChoosePMSet* to choose a set of models to pre-build (PM), subject to some maintenance cost constraint. We discuss query processing strategies using the PM's. Experiments on real world datasets demonstrate the effectiveness of our approaches and algorithms.

In the future, we plan to investigate query processing strategies with other prediction models. We also wish to investigate whether our techniques apply to multiple time series. Finally, developing visualization tools for users to "see the data" and to interact with the database system when exploring the model space is also an interesting research direction.

9. ACKNOWLEDGMENTS & REFERENCES

This work was supported by the NSF, under the grant IIS-0325838. We wish to thank the anonymous referees for several comments and suggestions that have improved the paper.

[1] Abraham, I., Aspnes, J., and Yuan, J. Skip B-trees. In *Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS 2005)*.

[2] T. Andersen, T. Bollerslev, and S. Lange. Forecasting financial market volatility: Sample frequency vis-à-vis forecast horizon. In *Journal of Empirical Finance*, Dec 1999, pages 457-477.

[3] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load Balancing and Locality in Range-Queriable Data Structures. In *Twenty-Third ACM Symposium on Principles of Distributed Computing*, pages 115–124, July 2004.

[4] James Aspnes and Gauri Shah. Skip Graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, January 2002.

[5] Boyer, C. B. Pascal's Formula for the Sums of Powers of the Integers. In *Scripta Math*. 9, 237-244, 1943.

[6] Brockwell, P., and Davis, R. *Introduction to Time Series and Forecasting*. 2nd Edition. Springer Texts in Statistics. 2002.

[7] Bulut, A. and Singh, A.K. SWAT: Hierarchical Stream Summarization in Large Networks. In *ICDE*, 2003.

[8] J. G. De Gooijer and R. J. Hyndman. 25 Years of IIF Time Series Forecasting: A Selective Review. June 2005. Tinbergen Institute Discussion Papers No. TI 05-068/4.

[9] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD* 2006.

[10] Duan S., and Babu, S. Processing Forecasting Queries. In *VLDB*, 2007.

[11] Eubank, R.L. *A Kalman Filter Primer*. Chapman & Hall/CRC, 2006.

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.

[13] D. Goldin and P. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In *CP'95,* Sep 1995.

[14] Hyndman, R.J., Kostenko, A.V. Minimum Sample Size Requirements for Seasonal Forecasting Models. In *Foresight*, Issue 6, Spring 2007.

[15] Makridakis, S., Wheelwright S., and Hyndman, R. Forecasting Methods and Applications. Third Edition. John Wiley & Sons, Inc. 1998.

[16] Mendenhall, W., and Sincich, T. Statistics for Engineering and the Sciences. Fourth Edition. Prentice-Hall, Inc. 1994.

[17] Mentzer, J.T., and J.E. Cox Jr. Familiarity, Application and Performance of Sales Forecasting Techniques. In *Journal of Forecasting*, 3, 1984, 27-36.

[18] M. Mitzenmacher, E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[19] J. Ian Munro, Thomas Papadakis, and Robert Sedgewick. Deterministic Skip Lists. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms (SODA'92)*. Orlando, Florida, United States. pp. 367-375.

[20] Leonore Neugebauer. Optimization and Evaluation of Database Queries Including Embedded Interpolation Procedures. In *SIGMOD*, 1991.

[21] Palpanas, T., Vlachos, M., Keogh, E., Gunopulos, D., and Truppel, W. Online amnesic approximation of streaming time series. In *ICDE*, 2004.

[22] Papadimitriou, S., Sun, J., and Yu, P. Local Correlation Tracking in Time Series. In *ICDM,* 2006.

[23] Papadimitriou, S., and Yu, P. Optimal Multi-scale Patterns in Time Series Streams. In *SIGMOD,* 2006.

[24] Pugh, W. Skip lists: a probabilistic alternative to balanced trees. In *Communications of the ACM*, June 1990, 33(6) 668-676.

[25] Schultz, H. J. The Sums of the *k*'th Powers of the First *n* Integers. In *Amer. Math. Monthly* 87, 478-481, 1980.

[26] J. Stewart. *Calculus: Concepts and Contexts (2nd ed.).* Thomson Learning, Inc. 2001.

[27] D. Tulone and S. Madden. PAQ: Time series forecasting for approximate query answering in sensor networks. In *EWSN*, 2006.

[28] Whitney, A., and Shasha, D. Lots o' Ticks: Real-time High Performance Time Series Queries on Billions of Trades and Quotes. In *SIGMOD*, 2001.

[29] Winklhofer, H., A. Diamantopoulos, and S.F. Witt. Forecasting Practice: A Review of the Empirical Literature and an Agenda for Future Research. In *International Journal of Forecasting*, 12, June 1996, 193-221.

[30] Wolniewicz, R., and Graefe, G. Algebraic Optimization of Computations over Scientific Databases. In *VLDB*, 1993.

[31] B. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online Data Mining for Co-evolving Time Sequences. In *ICDE*, 2000.

[32] Zhu, Y., and Shasha, D. Query by Humming: a Time Series Database Approach. In *SIGMOD*, 2003.

[33] http://dekorte.com/projects/opensource/SkipDB/.