Processing Forecasting Queries

Songyun Duan
Duke University
syduan@cs.duke.edu

Shivnath Babu*
Duke University
shivnath@cs.duke.edu

ABSTRACT

Forecasting future events based on historic data is useful in many domains like system management, adaptive query processing, environmental monitoring, and financial planning. We describe the Fa system where users and applications can pose declarative forecasting queries—both one-time queries and continuous queries—and get forecasts in real-time along with accuracy estimates. Fa supports efficient algorithms to generate execution plans automatically for forecasting queries from a novel plan space comprising operators for transforming data, learning statistical models from data, and doing inference using the learned models. In addition, Fa supports adaptive query-processing algorithms that adapt plans for continuous forecasting queries to the time-varying properties of input data streams. We report an extensive experimental evaluation of Fa using synthetic datasets, datasets collected on a testbed, and two real datasets from production settings. Our experiments give interesting insights on plans for forecasting queries, and demonstrate the effectiveness and scalability of our plan-selection algorithms.

1. INTRODUCTION

Forecasting future events based on historic data is applicable and useful in a range of domains like proactive system management, inventory planning, adaptive query processing, and sensor data management. On-demand computing systems, e.g., Amazon's Elastic Cloud [9], treat physical resources like servers and storage as a part of a shared computing infrastructure, and allocate resources dynamically to support changing application demands. These systems benefit significantly from early and accurate forecasts of workload surges and potential failures [13].

Adaptive query processing [3], where query plans and physical designs adapt to changes in data properties and resource availability, becomes very effective if these changes can be predicted with reasonable accuracy. Forecasting plays an important role in personal and enterprise-level decision making. For example, inventory planning involves forecasting future sales based on historic data; and day traders rapidly buy and sell stocks based on forecasts of stock performance [21].

In this paper, we describe the *Fa*[1] data-management system where users and applications can pose declarative *forecasting queries*. Fa supports efficient algorithms to generate execution plans for these queries, and returns forecasts and accuracy estimates in real-time. A forecasting query is posed over a multidimensional time-series dataset that represents historic data, as illustrated by the following example query.

Q_1: Select C
From Usage
Forecast 1 day

The full syntax and semantics of forecasting queries will be given in Section 2. The From clause in a forecasting query specifies the historic time-series data on which the forecast will be based. The query result will contain forecasts for the attributes listed in the Select clause, with the forecasts given for the timestamp(s) specified in the (new) Forecast clause. By default, this timestamp is specified as an interval, called *lead-time*, relative to the maximum timestamp in the historic data.

The time-series dataset "Usage" in example query Q_1 is shown in Figure 1(a). Usage contains daily observations from Day 5 to Day 17 of the bandwidth used on three links in an Internet Service Provider's network. Since Q_1 specifies a lead-time of 1 day, Q_1's result will be a forecast of attribute C for Day 18. Q_1 is a *one-time* query posed over a fixed dataset, similar to a conventional SQL query posed over relations in a database system. Our next example forecasting query Q_2 is posed as a *continuous* query over a windowed data stream.

Q_2: Select cpu_util, num_io, resp_time
From PerfMetricStream [Range 300 minutes]
Forecast 15 minutes, 30 minutes

Q_2 is expressed in the CQL continuous query language [1] extended with the Forecast clause. Here, "PerfMetricStream" is a continuous stream of performance metrics collected once every minute from a production database server. At timestamp τ (in minutes), Q_2 asks for forecasts of CPU utilization, number of I/Os, and response time for all timestamps in $[\tau + 15,\ \tau + 30]$. This forecast should be based on the window of data in PerfMetricStream over the most recent 300 minutes, i.e., tuples with timestamps in $[\tau - 299,\ \tau]$.

*Supported by an NSF CAREER award and gifts from IBM

VLDB '07, September 23-28, 2007, Vienna, Austria.

[1]African god of fate and destiny

Day	A	B	C
5	36	24	17
6	12	46	16
7	36	46	17
8	12	47	69
9	35	25	17
10	35	46	68
11	13	46	16
12	13	46	68
13	35	46	68
14	36	46	16
15	35	25	16
16	13	47	68
17	12	25	16

Day	A	B	C	A_{-1}	B_{-1}	C_{-1}	C_1
5	36	24	17	--	--	--	16
6	12	46	16	36	24	17	17
7	36	46	17	12	46	16	69
8	12	47	69	36	46	17	17
9	35	25	17	12	47	69	68
10	35	46	68	35	25	17	16
11	13	46	16	35	46	68	68
12	13	46	68	13	46	16	68
13	35	46	68	13	46	68	16
14	36	46	16	35	46	68	16
15	35	25	16	36	46	16	68
16	13	47	68	35	25	16	16
17	12	25	16	13	47	68	?

Day	A_{-2}	B_{-1}	C_1
5	--	--	16
6	--	24	17
7	36	46	69
8	12	46	17
9	36	47	68
10	12	25	16
11	35	46	68
12	35	46	68
13	13	46	16
14	13	46	16
15	35	46	68
16	36	25	16
17	35	47	?

Figure 1: Example datasets. (a) Usage; (b) and (c) are transformed versions of Usage

The problem we address in this paper is how to process forecasting queries automatically and efficiently in order to generate the most accurate answers possible based on patterns in the historic data; also giving accuracy estimates for forecasts. A forecasting query can be processed using an execution plan that builds and uses a statistical model from the historic data. The model captures the relationship between the value we want to forecast and the recent data available to make the forecast. Before building the model, the plan may apply a series of transformations to the historic data.

Let us consider a plan p to process our example query Q_1. Plan p first transforms the Usage dataset to generate the dataset shown in Figure 1(b). A tuple with timestamp τ (in days) in this transformed dataset contains the values of attributes A, B, and C for timestamps $\tau-1$ and τ from Usage, as well as the value of C for timestamp $\tau+1$. These figures use the notation X_δ to denote the attribute whose value at time τ is the value of attribute $X \in$ Usage at time $\tau+\delta$. Using the tuples from Day 6 to Day 16 in the transformed dataset as training samples, plan p builds a *Multivariate Linear Regression* (MLR) model [21] that can estimate the value of attribute C_1 from the values of attributes A, B, C, A_{-1}, B_{-1}, and C_{-1}. The MLR model built is:

$$C_1 = -0.7A - 1.04B - 0.43C - A_{-1} + 1.05B_{-1} - 0.32C_{-1} + 114.4$$

Once this model has been built, it can be used to compute Q_1's result, which is the "?" in Figure 1(b) because C_1 for Day 17 is equal to C for Day 18 given our transformation. By substituting $A = 12$, $B = 25$, $C = 16$, $A_{-1} = 13$, $B_{-1} = 47$, and $C_{-1} = 68$ from Day 17 (Figure 1(b)) into the MLR model, we get the forecast 87.78.

However, plan p has some severe shortcomings that illustrate the challenges in accurate and efficient forecasting:

1. It is nontrivial to pick the best set of transformations to apply before building the model. For example, if plan p had performed the appropriate attribute creation and removal to generate the transformed dataset shown in Figure 1(c), then the MLR model built from this data would forecast 64.10 which is more accurate than 87.78. Notice from the Usage data that when A_{-2} and B_{-1} are around 35 and 47 respectively, then C_1 is around 68.
2. Linear regression may fail to capture complex data patterns needed for accurate forecasting. For example, by building and using a *Bayesian Network* model on the dataset in Figure 1(c), the forecast can be improved to 68.5 (see Example 2 in Section 3). This observation raises a challenging question: how can we pick the best statistical model to use in a forecasting plan?
3. For high-dimensional datasets, most statistical models (including MLR) have very high model-building times, and often their accuracy degrades as well. This fact is problematic when forecasts are needed for real-time decisions, particularly in high-speed streaming applications. Furthermore, plans must adapt as old patterns disappear and new patterns emerge in time-varying streams.

1.1 Our Contributions and Outline

One-time and Continuous Forecasting Queries: In Section 2, we introduce the syntax and semantics of forecasting queries, showing how these queries can be expressed through simple extensions to conventional declarative languages for querying time-series datasets and data streams.

Plan Space: Section 3 describes a space of execution plans for forecasting queries. As illustrated above, a plan for forecasting is a combination of (i) *transformers* that transform the input data in desired ways, (ii) *builders* that generate statistical models from the transformed data, and (iii) *predictors* that make forecasts using the models generated.

Plan Selection: Sections 4 and 5 describe how to find a plan efficiently whose accuracy of forecasting is close to that of the best plan from the extremely large plan space for a forecasting query. The technical challenge comes from the fact that there are no known ways to estimate the accuracy of a plan without actually running the plan. Hence, conventional optimization frameworks based on cost models (e.g., [16]) are inapplicable here. Furthermore, running a plan is costly since statistical models can take a long time to build. In effect, we need algorithms that converge to fairly-accurate plans by running as few plans as possible.

Adaptive Query Processing: Section 8 describes how we process continuous forecasting queries over windowed data streams, where our plans need to adapt to the time-varying nature of the streams.

Experimental Evaluation: Sections 6 and 8 report an extensive experimental evaluation based on synthetic datasets, datasets collected on a testbed, and real operational datasets collected from two production settings (a popular web-site and a departmental cluster). This evaluation gives interesting insights on plans for forecasting queries, and demonstrates the effectiveness and scalability of our algorithms.

2. FORECASTING QUERIES

We consider multidimensional time-series datasets having a relational schema $\Gamma, X_1, X_2, \ldots, X_n$. Γ is a *timestamp* attribute with values drawn from a discrete and ordered domain $DOM(\Gamma)$. A dataset can contain at most one tuple for any timestamp $\tau \in DOM(\Gamma)$. Each X_i is a time series. We use $X_i(\tau)$ to denote X_i's value at time τ.

We defer the discussion of continuous forecasting queries over windowed data streams to Section 8. A one-time forecasting query over a fixed dataset has the general form:

Select *AttrList*
From D
Forecast [`absolute`] L [, [`absolute`] L']

$D(\Gamma, X_1, \ldots, X_n)$ is a time-series dataset, *AttrList* is a subset of $X_1, \ldots, X_n$, and L, L' are intervals or timestamps from $DOM(\Gamma)$. Terms enclosed within "[" and "]" are optional.

Before we consider the general case, we will first explain the semantics of a simple forecasting query "Select X_i From D Forecast L," which we denote as *Forecast*(D, X_i, L). Let D consist of m tuples with respective timestamps $\tau_1 < \tau_2 < \cdots < \tau_m$. Then, the result of *Forecast*(D, X_i, L) is the two-tuple $\langle X_i(\tau_m + L), acc\rangle$; $X_i(\tau_m + L)$ is the forecast and acc is the estimated accuracy of this forecast. Intuitively, the result of *Forecast*(D, X_i, L) is the forecast of attribute X_i for a timestamp that is L time units after the maximum timestamp in D; hence, L is called the lead-time of the forecast.

The extension to forecasting multiple attributes is straightforward. For example, the result of "Select X_i, X_j From D Forecast L," is a four-tuple $\langle X_i(\tau_m + L), acc_i, X_j(\tau_m + L), acc_j\rangle$, where acc_i and acc_j are the accuracy estimates of the $X_i(\tau_m + L)$ and $X_j(\tau_m + L)$ forecasts.

If the query specifies two lead-times L and L', then the query is called a *range forecasting query*, as opposed to a *point forecasting query* that specifies a single lead-time. The result of a range forecasting query contains a forecast and accuracy estimate for each specified attribute for each timestamp in the $[L, L']$ range. If the keyword `absolute` is specified before a lead-time L, then L is a treated as an absolute timestamp (e.g., March 1, 2007) rather than as an interval relative to the maximum timestamp in D.

Problem Setting: In this paper we consider point forecasting queries of the form *Forecast*(D, X_i, L). Range forecasting is not a straightforward extension of point forecasting, and will be considered in future work.

3. EXECUTION PLANS

A plan for a forecasting query contains three types of logical operators—*transformers*, *predictors*, and *builders*—and a summary data structure called *synopsis*.

- A *transformer* $T(D)$ takes a dataset D as input, and outputs a new dataset D' that may have a different schema from D.
- A *synopsis* $Syn(\{Y_1, \ldots, Y_N\}, Z)$ captures the relationship between attribute Z and attributes $Y_1, \ldots, Y_N$, such that a *predictor* $P(Syn, u)$ can use Syn to estimate the value of Z in a tuple u from the known values of $Y_1, \ldots, Y_N$ in u. Z is called Syn's *output attribute*, and $Y_1, \ldots, Y_N$ are called Syn's *input attributes*.
- A *builder* $B(D, Z)$ takes a dataset $D(\Gamma, Y_1, \ldots, Y_N, Z)$ as input and generates a synopsis $Syn(\{Y_1, \ldots, Y_N\}, Z)$.

Next, we give two example physical implementations each for the logical entities defined above.

Project transformer: A project transformer π_{list} retains attributes in the input that are part of the attribute list *list*, and drops all other attributes in the input dataset; so it is similar to a duplicate-preserving project in SQL.

Shift transformer: Shift(X_j, δ), where $1 \leq j \leq n$ and δ is an interval from $DOM(\Gamma)$, takes a dataset $D(\Gamma, X_1, \ldots, X_n)$ as input, and outputs dataset $D'(\Gamma, X_1, \ldots, X_n, X')$ where the newly-added attribute $X'(\tau) = X_j(\tau + \delta)$. When δ is positive (negative), then X' is copy of X_j that is shifted backward (forward) in time.

Example 1. The dataset in Figure 1(c) was computed from the Usage(A, B, C) dataset in Figure 1(a) by applying the transformers Shift$(A, -2)$, Shift$(B, -1)$, Shift$(C, 1)$, and the transformer $\pi_{A_{-2}, B_{-2}, C_1}$ in sequence. □

Multivariate Linear Regression (MLR): An *MLR synopsis* with input attributes $Y_1, \ldots, Y_N$ and output attribute Z estimates the value of Z as a linear combination of the Y_j values using the equation $Z = c + \sum_{j=1}^{N} \alpha_j Y_j$ [20]. The *MLR-builder* uses a dataset $D(\Gamma, Y_1, \ldots, Y_N, Z)$ to compute the *regression coefficients* α_j and the constant c. Note that this equation is actually a system of linear equations, one equation for each tuple in D. The MLR-builder computes the least-squares solution of this system of equations, namely, the values of α_js and c that minimize the sum of $(Z(\tau) - \hat{Z}(\tau))^2$ over all the tuples in D [21]. Here, $Z(\tau)$ and $\hat{Z}(\tau)$ are respectively the actual and estimated values of Z in the tuple with timestamp τ in D. Once all α_js and c have been computed, the *MLR-predictor* can estimate Z in a tuple given the values of attributes $Y_1, \ldots, Y_N$.

Bayesian Networks (BN): A *BN synopsis* is a summary structure that can represent the joint probability distribution $Prob(Y_1, \ldots, Y_N, Z)$ of a set of random variables $Y_1, \ldots, Y_N$, Z [20]. A BN for variables $Y_1, \ldots, Y_N, Z$ is a directed acyclic graph (DAG) with $N+1$ vertices corresponding to the $N+1$ variables. Vertex X in the BN is associated with a *conditional probability table* that captures $Prob(X|Parents(X))$, namely, the conditional probability distribution of X given the values of X's parents in the DAG. The DAG structure and conditional probability tables in the BN satisfy the following equation for all $(Y_1 = y_1, \ldots, Y_N = y_N, Z = z)$ [20]:

$$Prob(y_1, \ldots, y_N, z) = \prod_{i=1}^{N} Prob(Y_i = y_i | Parents(Y_i)) \times Prob(Z = z | Parents(Z))$$

Given a dataset $D(\Gamma, Y_1, \ldots, Y_N, Z)$, the *BN-builder* finds the DAG structure and conditional probability tables that approximate the above equation most closely for the tuples in D. Since this problem is NP-Hard, the BN-builder uses heuristic search over the space of DAG structures for $Y_1, \ldots, Y_N$, Z [20].

The *BN-predictor* uses the synopsis generated by the BN-builder from $D(\Gamma, Y_1, \ldots, Y_N, Z)$ to estimate the unknown value of Z in a tuple u from the known values of $u.Y_1, \ldots, u.Y_N$. The BN-predictor first uses the synopsis to infer the distribution $Prob(u.Z = z | u.Y_j = y_j, 1 \leq j \leq N)$. The exact value of $u.Z$ is then estimated from this distribution, e.g., by picking the expected value.

Example 2. Figure 2 shows the BN synopsis built by the BN-builder from the Usage(A_{-2}, B_{-1}, C_1) dataset in Figure 1(c). To compute example query Q_1's result, the BN-predictor will use the synopsis to compute $Prob(C_1 \in [16 - 17] | A_{-2} = 35, B_{-1} = 47)$ and $Prob(C_1 \in [68 - 69] | A_{-2} = 35, B_{-1} = 47)$, which are 0 and 1 respectively; hence the forecast 68.5 will be output. □

3.1 Initial Plan Space Considered (Φ)

There is a wealth of possible synopses, builders, predictors, and transformers from the statistical machine-learning

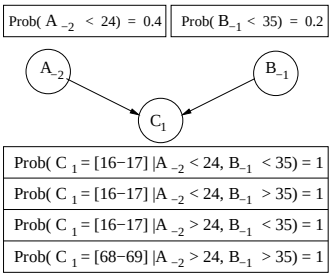

Figure 2: BN synopsis built from data in Fig. 1(c)

literature [20]. For clarity of presentation, we begin by considering a focused physical plan space Φ. Section 7 describes how our algorithms can be applied to larger plan spaces.

An execution plan $p \in \Phi$ for a *Forecast*(D, X_i, L) query first applies a sequence of transformers to D, then uses a builder to generate a synopsis from the transformed dataset, and finally uses a predictor to make a forecast based on the synopsis and the transformed dataset. For example, the transformers in Example 1, followed by the BN synopsis built by the BN-builder and used to make the forecast by the BN-predictor in Example 2, is one complete plan in Φ.

Transformers: The transformers in a plan $p \in \Phi$ are limited to Shift and π which play a critical role in presenting the input data in ways that a synopsis built from the data can capture patterns useful for forecasting. Shift creates relevant new attributes—not present in the original input dataset—to include in a synopsis. π eliminates irrelevant and redundant attributes that harm forecasting if included in synopses: (i) irrelevant attributes can reduce forecasting accuracy by obscuring relevant patterns, and (ii) redundant attributes can increase the time required to build synopses.

Synopses, Builders, and Predictors: A plan $p \in \Phi$ contains one of five popular synopses—along with the builder and predictor for that synopsis—from the machine-learning literature: Multivariate Linear Regression (MLR), Bayesian Networks (BN), *Classification and Regression Trees* (*CART*), *Support Vector Machines* (*SVM*), and *Random Forests* (*RF*) [20]. The choice of which synopses to include in Φ was guided by some recent studies that compare various synopses; see Section 9 for details. MLR and BN are described in Section 3. We give very brief descriptions of the other synopses below; more details, including descriptions of respective builders and predictors, are in the technical report [8]. It is not necessary to understand the specific details of these synopses to understand our contributions.

- **CART** synopses capture attribute relationships as a decision tree where the nonleaf nodes check conditions and the leaf nodes forecast values.
- **SVM** synopses map training data points into a high-dimensional space and determine linear hyperplanes that best separate (classify) the data.
- **RF** synopses learn many CARTs by choosing attributes randomly from a dataset, and combine forecasts from individual CARTs. RFs represent *ensemble learning* [20] that is used widely in machine-learning today.
- **Time-Series Methods**: In conjunction with Shift transformers, MLR synopses can capture popular time-series forecasting methods like univariate *autoregression* [5] and multivariate *Muscles* [21].

4. PLAN-SELECTION PRELIMINARIES

In this section we describe the important characteristics of the problem of selecting a good plan from Φ for a *Forecast*(D, X_i, L) query. We describe the structure of plans from Φ, estimate the total size of the plan space, and show the difficulty in estimating the forecasting accuracy of a plan.

Observation 1. **(Plan structure)** Let $D(\Gamma, X_1, \ldots, X_n)$ be a time-series dataset. A plan $p \in \Phi$ for a *Forecast*(D, X_i, L) query can be represented as $\langle Y_1, \ldots, Y_N, type, Z\rangle$ where:

- $type \in$ {MLR,BN,CART,SVM,RF} is p's synopsis type. The synopsis type uniquely determines both p's builder and p's predictor.
- $Y_1, \ldots, Y_N$ are attributes such that each $Y_i = X_j(\tau + \delta)$ for some j and δ, $1 \leq j \leq n$ and $\delta \leq 0$. $Y_1, \ldots, Y_N$ are the input attributes of p's synopsis.
- Attribute $Z = X_i(\tau + L)$ is the output attribute of p's synopsis.

This observation can be justified formally based on our assumptions about Φ and the definitions of synopses, builders, predictors, and transformers. Because of space constraints, we provide an intuitive explanation only.

Recall that plan p for *Forecast*(D, X_i, L) will use a synopsis to estimate the query result, namely, the value of $X_i(\tau_m + L)$, where τ_m is the maximum timestamp in D. p's synopsis has access only to data in D up to timestamp τ_m in order to estimate $X_i(\tau_m + L)$. Furthermore, Φ has only the Shift transformer to create new attributes apart from attributes $X_1, \ldots, X_n$. Given these restrictions, the input attributes in p's synopsis can only be $X_j(\tau + \delta)$, $1 \leq j \leq n$ and $\delta \leq 0$, and the output attribute is $X_i(\tau + L)$.

Observation 2. **(Size of plan space)** Suppose we restrict the choice of Shift transformers to derive input attributes for synopses in a plan to Shift(X_j, δ), $1 \leq j \leq n$ and $-\Delta \leq \delta \leq 0$. Then, the number of unique plans in Φ, $|\Phi| = 5 \times 2^{n(1+\Delta)}$.

In one of our application domains, $n = 252$ and $\Delta = 90$, so $|\Phi| = 5 \times 2^{22932}$!

4.1 Estimating Forecasting Accuracy of a Plan

The *forecasting accuracy* (accuracy) of a plan $p = \langle Y_1, \ldots, Y_N, type, Z\rangle$ is the accuracy with which p's synopsis can estimate the true value of Z given the values of $Y_1, \ldots, Y_N$. The goal of plan selection for a $Q =$*Forecast*(D, X_i, L) query is to find a plan that has accuracy close to the best among all plans for Q in Φ. To achieve this goal, we need a way to compute the accuracy of a given plan.

The preferred technique in statistical machine-learning to estimate the accuracy of a synopsis is called *K-fold cross-validation* (*K-CV*) [20]. K-CV can estimate the accuracy of a plan p that builds its synopsis from the dataset $D(\Gamma, Y_1, \ldots, Y_N, Z)$. K-CV partitions D into K (nonoverlapping) partitions, denoted $D_1, \ldots, D_K$. (Typically, $K = 10$.) Let $D'_i = D - D_i$. For $i \in [1, K]$, K-CV builds a synopsis $Syn_i(\{Y_1, \ldots, Y_N\}, Z)$ using the tuples in D'_i, and uses this synopsis to estimate the value of $u.Z$ for each tuple $u \in D_i$. This computation will generate a pair $\langle a_j, e_j\rangle$ for each of the m tuples in D, where a_j is the tuple's actual value of Z and e_j is the estimated value. Any desired accuracy metric can be computed from these pairs, e.g., *root mean squared error* $= \sqrt{\sum_{j=1}^{m} \frac{(a_j - e_j)^2}{m}}$, giving an accuracy estimate for p.

Observation 3. **(Estimating accuracy)** K-CV is a robust technique to estimate the accuracy of a plan $p = \langle Y_1,$

Algorithm *Fa's Plan Search (FPS)*
Input: *Forecast*$(D(\Gamma, X_1, \ldots, X_n), X_i, L)$ query
Output: Forecasts and accuracy estimates are output as FPS runs. FPS is terminated when a forecast with satisfactory accuracy is obtained, or when lead-time runs out;
1. $l = 0$; /* current iteration number of outer loop */
BEGIN OUTER LOOP
2. Attribute set $Attrs = \{\}$; /* initialize to an empty set */
3. $l = l + 1$; /* consider Δ more shifts than in last iteration */
4. **FOR** $j \in [1, n]$ and $\delta \in [-l\Delta, 0]$
   Add to *Attrs* the attribute created by Shift(j, δ) on D;
5. Generate attribute $Z = X_i(\tau + L)$ using Shift(X_i, L) on D;
6. Let the attributes in *Attrs* be $Y_1, \ldots, Y_q$. Rank $Y_1, \ldots, Y_q$ in decreasing order of relevance to Z; /* Section 5.2 */
/* Traverse the ranked list of attributes from start to end */
   BEGIN INNER LOOP
7. Pick next chunk of attributes from list; /* Section 5.3 */
8. Decide whether a complete plan should be generated using the current chunk of attributes; /* Section 5.5 */
9. **IF Yes**, find the best plan p that builds a synopsis using attributes in the chunk. If p has the best accuracy among all plans considered so far, output the value forecast by p, as well as p's accuracy estimate; /* Section 5.4 */
   END INNER LOOP
END OUTER LOOP

Figure 3: Plan selection and execution algorithm for one-time forecasting queries

$\ldots, Y_N, type, Z\rangle$ without knowing the actual query result. However, K-CV builds a synopsis of type *type* K times, and uses these synopses to estimate Z for each input tuple.

In effect, K-CV is computationally expensive. However, discussions with researchers in statistical machine-learning revealed that there are no known techniques to estimate the accuracy of an MLR, BN, CART, SVM, or RF synopsis fairly accurately on a dataset D without actually building the synopsis on D. (In Section 6.6, we report experimental results related to this point.) Therefore, K-CV is as good a technique as any to estimate the accuracy of a plan, and we will use it for that purpose.

5. PROCESSING ONE-TIME QUERIES

The observations in Section 4 motivate our algorithm, called *Fa's Plan Search (FPS)*, to process a one-time forecasting query *Forecast*(D, X_i, L) query.

5.1 Overview of FPS

Figures 3 and 4 give an overview of FPS which runs as a two-way nested for loop. For simplicity of presentation, Figure 3 does not show how computation can be shared within and across the loops of FPS; sharing and other scalability issues are discussed in Section 5.6.

FPS's outer loop enumerates a large number of attributes of the form $X_j(\tau + \delta)$, $1 \leq j \leq n$ and $-l\Delta \leq \delta \leq 0$, where $\Delta \geq 0$ is a user-defined constant. The value of l is increased across iterations of the loop so that more and more attributes will be enumerated progressively. FPS aims to pick subsets of attributes from the enumerated set such that synopses built from these subsets give good accuracy. Intuitively, a combination of a highly-predictive attribute subset $Y_1, \ldots, Y_N$ and an appropriate synopsis type *type* gives a plan $p = \langle Y_1, \ldots, Y_N, type, Z\rangle$ for estimating $Z = X_i(\tau + L)$ with good accuracy.

As illustrated in Figure 4 and Line 6 of Figure 3, FPS ranks the enumerated attributes in decreasing order of relevance to attribute Z. Techniques for ranking are described in Section 5.2. In its inner loop, FPS traverses the ranked list from highly relevant to less relevant attributes, and extracts one chunk of attributes at a time. Techniques for traversing the ranked list are described in Section 5.3.

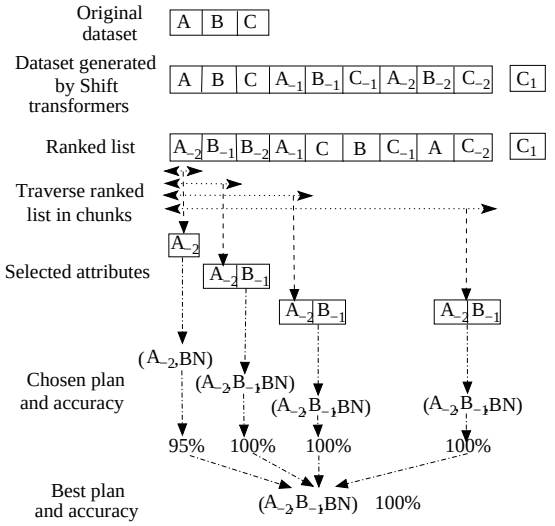

Figure 4: Pictorial view of FPS processing Example query Q_1 from Section 1

For each chunk C, FPS decides whether or not to derive a complete plan from C. Section 5.5 describes how this decision is made, and the implications of making a wrong decision. If the decision is to derive a plan, then FPS finds a highly-predictive subset of attributes $Y_1, \ldots, Y_N$ from C, and an appropriate synopsis type *type*, to produce the best plan $p = \langle Y_1, \ldots, Y_N, type, Z\rangle$ possible from C. These techniques are described in Section 5.4. Recall from Section 4.1 that FPS has to build p's synopsis in order to estimate p's accuracy. Once the synopsis is built, the extra cost to use the synopsis to compute the value forecast by p is small.

Whenever FPS finds a plan p whose accuracy estimate is better than the accuracy estimates of all plans produced so far, FPS outputs the value forecast by p as well as p's accuracy estimate. Thus, FPS produces more and more accurate forecasts in a progressive fashion, similar to online aggregation techniques proposed for long-running SQL queries [11]. FPS runs until it is terminated by the application/user who issued the query, e.g., a user may terminate FPS when she is satisfied with the current accuracy estimate.

We now present the options we considered to implement each step in FPS. These options are evaluated experimentally in Section 6, and a concrete instantiation of FPS is presented in Section 7.

5.2 Ranking Attributes

Line 6 in Figure 3 ranks the attributes enumerated by FPS's outer loop in decreasing order of relevance to the output attribute Z. Intuitively, attributes more relevant to Z are more likely to give good forecasts when included in a synopsis to forecast Z. Such relevance can be computed in one of two ways: (i) *correlation-based*, where attributes are ranked based on their correlation with Z; and (ii) *time-based*, where attributes whose values are more recent are ranked higher.

Correlation-based Ranking: There exist two general approaches to measure correlation between attributes Y and Z,

one based on linear-correlation theory and the other based on information theory [22]. *Linear correlation coefficient* (*LCC*) is a popular measure of linear correlation [20].

$$LCC(Y,Z) = LCC(Z,Y) = \frac{\sum_i (y_i - \overline{Y})(z_i - \overline{Z})}{\sqrt{\sum_i (y_i - \overline{Y})^2}\sqrt{\sum_i (z_i - \overline{Z})^2}}$$

Each (y_i, z_i) is a pair of (Y, Z) values in a tuple in the input dataset, and $\overline{Y}$ and $\overline{Z}$ denote the respective means. LCC is efficient to compute, but it may not capture nonlinear correlation.

Information gain of Z given Y, denoted $IG(Z,Y)$, is an information-theoretic measure of correlation between Y and Z computed as the amount by which the *entropy* of Z decreases after observing the value of Y. $IG(Z,Y) = H(Z) - H(Z|Y) = H(Y) - H(Y|Z) = IG(Y,Z)$, where H denotes entropy, e.g., $H(Y) = -\sum_i Prob(y_i)\log_2(Prob(y_i))$.

$$IG(Z,Y) \;=\; IG(Y,Z) = -\sum_i Prob(z_i)\log_2(Prob(z_i)) + \sum_j Prob(y_j)\sum_i Prob(z_i|y_j)\log_2(Prob(z_i|y_j))$$

Information gain is biased towards attributes with many distinct values, so a normalized variant called *symmetrical uncertainty* (*SU*) is often used to measure correlation.

$$SU(Y,Z) = SU(Z,Y) = 2\left[\frac{IG(Y,Z)}{H(Y)+H(Z)}\right] \quad (1)$$

Time-based Ranking: Time-based ranking assumes that an attribute whose value was collected closer in time to τ is likely to be more predictive of $Z(\tau)$ than an attribute whose value was collected earlier in time. For example, consider attributes Y_1 and Y_2 created from the input dataset D by Shift(X_1, δ_1) and Shift(X_2, δ_2) transformers respectively. Because of the shift, the value of $Y_1(\tau)$ ($Y_2(\tau)$) comes from a tuple in D with timestamp $\tau + \delta_1$ ($\tau + \delta_2$). Let $\delta_1 < \delta_2 \leq 0$. Then, $Y_2(\tau)$ is more relevant to $Z(\tau)$ than $Y_1(\tau)$.

5.3 Traversal of the Ranked List in Chunks

Single Vs. Multiple Chunks: FPS traverses the ranked list in multiple overlapping chunks $C_1 \subset C_2 \subset \cdots \subset C_k$ that all start from the beginning of the list (see Figure 4). If all the attributes are considered as a single chunk (i.e., if $k = 1$), then the time for attribute selection and synopsis learning can be high because these algorithms are nonlinear in the number of attributes. Overlapping chunks enable FPS to always include the highly relevant attributes that appear at the beginning of the list. The potential disadvantage of overlapping chunks is the inefficiency caused by repetition of computation when the same attributes are considered multiple times. However, as we will show in Section 5.6, FPS can share computation across overlapping chunks, so efficiency is not compromised.

Fixed-size Vs. Variable-size Increments: Successive overlapping chunks $C_1 \subset C_2 \subset \cdots \subset C_k$ can be chosen with fixed-size increments or variable-size increments. An example of consecutive chunk sizes with fixed-size increments is $|C_1| = 10, |C_2| = 20, |C_3| = 30, |C_4| = 40$ and so on (arithmetic progression), while an example with variable-size increments is $|C_1| = 10, |C_2| = 20, |C_3| = 40, |C_4| = 80$ and so on (geometric progression).

5.4 Generating a Plan from a Chunk

At Line 9 in Figure 3, we need to generate a complete plan from a chunk of attributes C. This plan $\langle \Omega, type, Z \rangle$ should contain attributes $\Omega \subseteq C$ and a synopsis type *type* that together give the best accuracy in estimating the output attribute Z among all choices for $(\Omega, type)$. We break this problem into two subproblems: (i) finding an attribute subset $\Omega \subseteq C$ that is highly predictive of Z, and (ii) finding a good synopsis type for Ω.

5.4.1 Selecting a Predictive Attribute Subset

The problem of selecting a predictive attribute subset $\Omega \subseteq C$ can be attacked as a search problem where each state in the search space represents a distinct subset of C [10]. Since the space is exponential in the number of attributes, heuristic search techniques can be used. The search technique needs to be combined with an estimator that can quantify the predictive ability of a subset of attributes. We consider three methods for attribute selection:

1. Wrapper [10] estimates the predictive ability of an attribute subset Ω by actually building a synopsis with Ω as the input attribute set, and using K-CV to estimate accuracy. Wrapper uses *Best First Search* (*BFS*) [20] as its heuristic search technique. BFS starts with an empty set of attributes and enumerates all possible single attribute expansions. The subset with the highest accuracy estimate is chosen and expanded in the same manner by adding single attributes. If no improvement results from expanding the best subset, up to k (usually, $k = 5$) next best subsets are considered. The best subset found overall is returned when the search terminates. Building synopses for all enumerated subsets makes Wrapper good at finding predictive subsets, but computationally expensive.

2. Correlation-based Feature Selection (CFS) [10] is based on the heuristic that a highly-predictive attribute subset Ω is composed of attributes that are highly correlated with the (output) attribute Z, yet the attributes in Ω are uncorrelated with each other. This heuristic gives the following estimator, called *CFS Score*, to evaluate the ability of a subset Ω, containing k (input) attributes $Y_1, \ldots, Y_k$, to predict the (output) attribute Z (Equation 1 defines SU):

$$CFS\ Score(\Omega) = \frac{\sum_{i=1}^{k} SU(Y_i, Z)}{\sqrt{k + \sum_{i=1}^{k}\sum_{j\neq i, j=1}^{k} SU(Y_i, Y_j)}} \quad (2)$$

The numerator in Equation 2 is proportional to the input-output attribute correlation, so it captures how predictive Ω is of Z. The denominator is proportional to the input-input attribute correlation within Ω, so it captures the amount of redundancy among attributes in Ω. CFS uses BFS to find the attribute subset $\Omega \subseteq C$ with the highest CFS Score.

3. Fast Correlation-based Filter (FCBF) [22] is based on the same heuristic as CFS, but it uses an efficient deterministic algorithm to eliminate attributes in C that are either uncorrelated with the output attribute Z, or redundant when considered along with other attributes in C that have higher correlation with Z. While CFS's processing-time is usually proportional to n^2 for n attributes, FCBF's processing-time is proportional to $n \log n$.

5.4.2 Selecting a Synopsis Type

To complete a plan from an attribute subset Ω, we need to pick the synopsis type that gives the maximum accuracy in estimating Z using Ω. Recall from Section 4.1 that there are no known techniques to compute the accuracy of a synopsis without actually building the synopsis. One simple, but expensive, strategy in this setting is to build all five synopsis types, and to pick the best one. As an alternative,

Parameter/Option name	Default
Forecast lead time (Sec. 2)	25
Δ in FPS (Fig. 3)	90
Ranking attributes (Sec. 5.2)	Correlation-based (LCC)
Ranked-list traversal (Sec. 5.3)	Variable-sized (10×2^i)
Attribute selection (Sec. 5.4)	FCBF
Synopsis type (Sec. 5.4)	BN
Generate plan or not? (Sec. 5.5)	No pruning
Sharing across chunks (Sec. 5.6)	FPS-incr-cum

Table 1: Defaults for experiments

we tried to determine experimentally whether one or more synopsis types in BN, CART, MLR, SVM, and RF compare well in general to the other types in terms of both the accuracy achieved and the time to build. The results are very encouraging, and are reported in Section 6.5.

5.5 Decision to Generate a Plan or Not

At Line 8 in Figure 3, we need to decide whether to generate a complete plan using the current chunk of attributes C. Generating a complete plan p from C involves an attribute selection and building one or more synopses, so it is desirable to avoid or reduce this cost if p is unlikely to be better than the current best plan. To make this decision efficiently, we need an estimator that gives a reasonably-accurate and efficiently-computable estimate of the *best accuracy possible* from chunk C. We can *prune* C if the estimated best accuracy from C is not significantly higher than the accuracy of the current best plan. We considered two options:

- **No Pruning**, where a plan is generated for all chunks.
- **Pruning(k)**, where chunk C is pruned if the best CFS Score (Equation 2) among attribute subsets in C is worse than the k-th best CFS score among all chunks so far.

5.6 Sharing Computation in FPS

Recall from Section 5.3 that FPS traverses the ranked list of attributes in overlapping chunks $C_1 \subset C_2 \subset \cdots \subset C_k$ starting from the beginning of the list. The version of FPS described in Figure 3—which we call *FPS-full*—performs attribute selection and synopsis building from scratch for each chunk where a complete plan is generated.

An incremental version of FPS, called *FPS-incr*, shares (or reuses) computation across chunks by generating a plan from the attributes $\Theta \cup \{C_i - C_{i-1}\}$ at the ith chunk, instead of generating the plan from C_i. Here, $C_i - C_{i-1}$ is the set of attributes in C_i that are not in C_{i-1}, and $\Theta \subseteq C_{i-1}$ enables reuse of computation done for previous chunks. We consider two options for choosing Θ:

$$\Theta = \begin{cases} \Theta_{i-1} & (\textit{FPS-incr-cum}) \\ \Theta_{j'}, \quad j' = \text{argmax}_{1 \le j \le i-1}\, acc_j & (\textit{FPS-incr-best}) \end{cases}$$

Here, $\Theta_j \subseteq C_j$ is the attribute subset chosen in the plan for chunk C_j (Section 5.4.1), with corresponding accuracy acc_j. Intuitively, Θ in FPS-incr-cum tracks the cumulative attribute subset selected so far, while FPS-incr-best uses the best attribute subset among all chunks so far.

Note that sharing can be done across chunks in a ranked list—like we described above—as well as across the much larger overlapping chunks of attributes considered by successive iterations of FPS's outer loop; our description applies to this case as well.

6. EXPERIMENTAL EVALUATION

Our algorithms for processing one-time and continuous forecasting queries have been implemented in the Fa data management system being developed at Duke. Fa is targeted at managing the massive volumes of high-dimensional data generated by system and database monitoring applications. Fa's implementation of all synopses is based on the open-source *WEKA* toolkit [20]. Table 1 indicates the experimental defaults.

6.1 Datasets, Queries, and Balanced Accuracy

Since Fa's target domain is system and database monitoring, the datasets, forecasting queries, and accuracy metrics we present in this paper are drawn from this domain. The complete details of our experimental evaluation—including results omitted due to lack of space and datasets from other application domains—are given in the technical report [8]. We used real, testbed, and synthetic datasets; described next and in Table 2.

Real datasets: We consider two real datasets: *Aging-real* and *FIFA-real*. Aging-real is a record of OS-level data collected over a continuous period of two months from nine production servers in the Duke EE departmental cluster; [18] gives a detailed analysis. The predictive patterns in this dataset include the effects of *software aging*—progressive degradation in performance due to, e.g., memory leaks, unreleased file locks, and fragmented storage space—causing transient system failures. We use Aging-real to process queries that forecast with lead-time L whether a server's average response time will exceed a specified threshold.

FIFA-real is derived from a 92-day log for the 1998 FIFA Soccer World Cup web-site; [2] gives a detailed analysis. We use this dataset to process queries that forecast with lead-time L whether the web-site load will exceed a specified threshold. The load on the FIFA web-site is characteristic of most popular web-sites—with periodic segments, high burstiness at small time scales, but more predictability at larger time scales, and occasional traffic spikes.

Testbed datasets: These datasets contain OS, DBMS, and transaction-level performance metrics collected from a MySQL DBMS running an OLTP workload on a monitoring testbed we have developed. Table 2 gives brief descriptions. These datasets are used to process queries that forecast with lead-time L whether the average transaction response time will exceed a specified threshold. Our testbed datasets cover (i) periodic workloads—with different period lengths and complexity of pattern—(ii) aging behavior—both periodic and non-periodic, fixed and varying rate of aging—and (iii) multiple performance problems (e.g., CPU and I/O contention) happening in overlapping and nonoverlapping ways.

Synthetic datasets: We also generated synthetic time-series datasets to study the robustness of our algorithms; see Table 2.

Balanced Accuracy (BA): Notice that the queries we consider in our experimental evaluation forecast whether the system will experience a performance problem L time units from now. The accuracy metric preferred for such queries in the system management domain is called *Balanced Accuracy (BA)* [13]. BA is computed using K-CV (Section 4.1) as: BA = 0.5(1 - FP) + 0.5(1 - FN). Here, FP is the ratio of false positives (predicting a problem when there is none) in the ⟨actual value, estimated value⟩ pairs. Similarly, FN is the ratio of false negatives (failing to predict an actual problem).

Graphs: The majority of our graphs track the progress of FPS over time; the X axis shows the elapsed time since the query was submitted, and the Y axis shows the best BA among plans generated so far. Fast convergence to the

Name	n	m	I	Description
1. Aging-real	44	4820	15	OS-level data collected for 55 days from nine Duke EE departmental servers
2. FIFA-real	124	2068	60	Load metrics collected for 92 days from the 1998 Soccer World Cup web-site
3. Periodic-small-tb	196	4315	1	10-minute period; query parameter values varied to vary transaction response time
4. Periodic-large-tb	252	6177	1	90-minute period; resource contention created by varying number of DBMS threads
5. Aging-fixed-tb	252	1499	1	non-periodic; resource contention caused by an aging [18] CPU-intensive thread
6. Multi-large-tb	252	5000	1	non-periodic; resource contention caused by both CPU and disk contention
7. Multi-small-tb	252	719	1	16-minute period; resource contention caused by both CPU and disk contention
8. Aging-variant-syn	3	7230	1	non-periodic; rate of aging is not fixed like in Aging-fixed-tb
9. Complex-syn	6	14760	1	non-periodic; pattern simulating a problem that affects response time with a lag

Table 2: Datasets used in experiments; n, m, and I are the number of attributes, tuples, and measurement interval (minutes) respectively; real, tb, and syn represent real, testbed, and synthetic datasets respectively

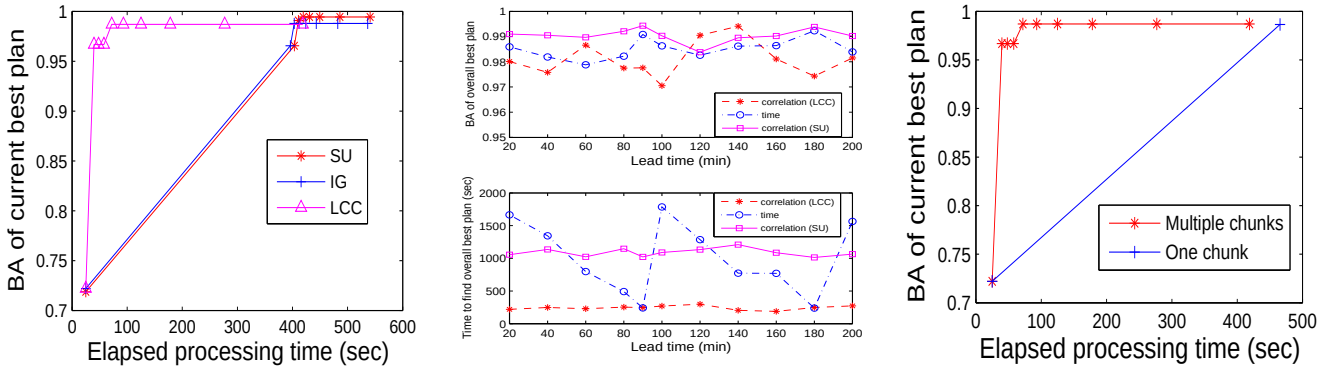

Figure 5: (a) Comparing correlation metrics, (b) Correlation Vs. time-based ranking, (c) Importance of chunk-based traversal. All three experiments use the Periodic-large-tb dataset.

maximum BA possible indicates good performance. BA is always ≥ 0.5, so the minimum Y value in these graphs is 0.5.

6.2 Ranking Attributes

Recall from Section 5.2 that FPS can choose to do time-based ranking of enumerated attributes, or correlation-based ranking using LCC, IG, or SU. Figures 5(a) and 5(b) show the general trend that we have seen across all our datasets and queries (note that the full set of performance graphs from our evaluation is given in the technical report [8]):

- FPS using LCC converges much faster and to almost the same BA as SU and IG. The worst results for LCC were for the synthetic Complex-syn dataset where we forced the predictive attributes to have nonlinear correlation with the output attribute Z. Even in this case, LCC converged to the same BA as IG and SU, with 20% more time than SU and 14% more time than IG.
- Correlation-based ranking using LCC is more robust than time-based ranking. Note the sensitivity of time-based ranking to the lead-time in Figure 5(b): time-based ranking will converge quickly only when the predictive patterns are "close by" in time, which *depends on the lead-time* for Periodic-large-tb.

6.3 Traversal of Ranked List

The trend in Figure 5(c) was consistent across all our datasets: it is significantly better to consider attributes in the ranked list in multiple overlapping chunks rather than considering all attributes all-at-once. The main reason is that the time for attribute selection and synopsis learning is nonlinear in the number of attributes. With multiple overlapping chunks, the chunks are smaller since the effects of computation sharing from Section 5.6 kick in.

We have found variable-size traversal (Section 5.3)—with chunk sizes increasing in a geometric progression 10×2^i up to a maximum size, then switching to an arithmetic progression with the last increment—to be more effective than fixed-size traversal:

- If ranking correctly brings the most predictive attributes to the beginning of the ranked list, then FPS gets close to the best accuracy from the first few chunks. Here, fixed-size and variable-size traversal are comparable.
- If the predictive attributes are not at the beginning of the ranked list—e.g., because the use of LCC for ranking failed to capture nonlinear correlation, or Δ was too low—then FPS may not get good accuracy until the middle/last chunks are considered or until more attributes are enumerated; variable-size traversal can get there sooner. This effect is clear in Figure 6 which considers a case where $\Delta = 30$ was low, so multiple iterations of the outer loop in Figure 3 were required to get to the best BA possible. Note the log scale on the X axis.

6.4 Selecting a Predictive Attribute Subset

Figure 7 represents the consistent trend we observed: (i) The running times of the attribute-selection techniques are in the order FCBF < CFS $\ll$ Wrapper, with Wrapper being about an order of magnitude slower than FCBF and CFS, without any significant advantage in accuracy; (ii) FCBF tends to perform better on average than CFS, but other features of FPS—mainly, sharing of computation (Section 5.6)—blur the difference between FCBF and CFS. (*Principal Component Analysis* also performs worse than FCBF/CFS.)

6.5 Selecting a Synopsis

Table 3 shows the time to produce the best plan, and the corresponding BA, for FPS when using BN, CART, MLR, and SVM synopses. As a comparison point, we provide the best performance of RF synopses which were found to be one of the best synopses available today in a recent comprehensive study [4]. (More detailed comparisons with RFs are provided in Section 7.)

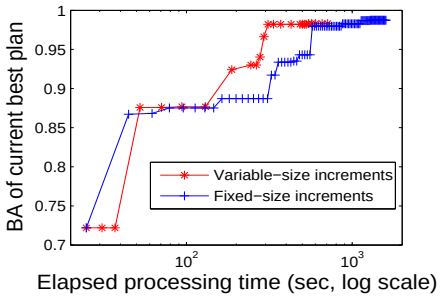

Figure 6: Choosing chunk sizes (Periodic-large-tb, Δ=30 in Fig. 3)

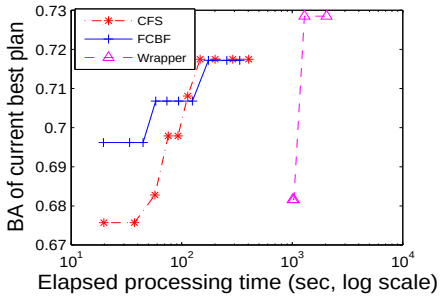

Figure 7: Comparing attribute-selection techniques (Aging-real)

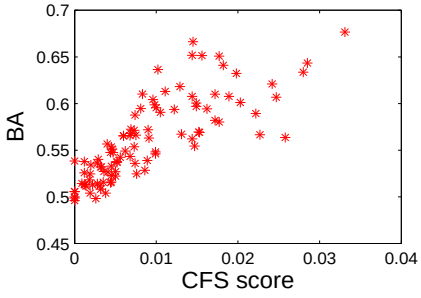

Figure 8: CFS-Score Vs. BA for random subsets (Multi-large-tb)

	FPS (BN)		FPS (CART)		FPS (MLR)		FPS (SVM)		RF	
Dataset	BA	Time	BA	Time	BA	Time	BA	Time	BA	Time
Aging-real	0.71	62.4	0.71	134.9	0.64	35.8	0.51	1948.7		
FIFA-real	0.87	28.8	0.85	36.6	0.84	201.4				
Periodic-small-tb	0.84	44.9	0.85	249.3	0.80	130.3			0.86	22339.7
Periodic-large-tb	0.98	71.98	0.98	167.7	0.97	92.6				
Aging-fixed-tb	0.91	27.5	0.93	56.8	0.89	145.3				
Multi-large-tb	0.72	99.7	0.73	197	0.71	100.8				
Multi-small-tb	0.91	53.2	0.91	49.7	0.85	19.1	0.86	482.3	0.91	933.2
Aging-variant-syn	0.82	14.2	0.81	109.4	0.80	24.2			0.85	3200.1
Complex-syn	0.99	130.1	0.99	506.4	0.99	134.7				

Table 3: Synopsis comparisons. The time in seconds to produce the best plan, and the corresponding BA, are shown. The missing data points are cases where Java ran out of heap space before convergence.

- The running times for SVM and RF are consistently worse than for BN/CART/MLR. The missing data points for SVM and RF are cases where Java runs out of heap space on our machine before the algorithms converge; showing the high memory overhead of these algorithms.
- BN and CART are competitive with other synopses in terms of both accuracy and running time; we attribute this performance to the fact that once the right transformations are made, a reasonably-sophisticated synopsis can attain close to the best accuracy possible.

6.6 Decision to Generate a Plan or Not

Our attempt to use the best CFS Score from a set of attributes Ω as an estimator of the best BA from Ω gave mixed results. Figures 8 and 9 plot the CFS Score and corresponding BA for a large set of randomly chosen attribute subsets from Multi-large-tb and Periodic-large-tb: CFS Score seems to be a reasonable indicator of BA for Multi-large-tb (note the almost linear relationship), but not so for Periodic-large-tb (note the points where CFS Score is reasonably high, but BA is far below the best). Figure 10 shows the performance of CFS-Score-based pruning, for k=1 and k=5, for Periodic-large-tb. While pruning does sometimes reduce the total processing-time considerably, it does not translate into faster convergence; in fact, convergence can be delayed significantly as seen in Figure 10.

6.7 Effect of Sharing

Figure 11 shows the consistent trend we observed across all datasets: computation sharing improves the speed of convergence of FPS significantly, without any adverse effect on accuracy. Note that sharing can be done across chunks in a ranked list as well as across the much larger chunks of attributes considered by successive iterations of FPS's outer loop. To show both effects, we set $\Delta = 30$ instead of the default 90 in Figure 11, so multiple iterations of the outer loop are required to converge.

7. MAKING FPS CONCRETE

The trends observed in our experimental evaluation point to reasonable defaults for each step of FPS in Figure 3, to find a good plan quickly from Φ for a one-time *Forecast*(D, X_i, L) query:

- Use LCC for ranking attributes.
- Traverse the ranked list in multiple overlapping chunks, with increasing increments in chunk size up to a point.
- Generate a complete plan from each chunk considered.
- Use FCBF or CFS for attribute selection.
- Build BN or CART synopses.
- Use FPS-incr-cum for computation sharing.

A recent comprehensive study [4] found RFs to be one of the best synopses available today. Figure 12 compares the above concrete instantiation of FPS with:

- *RF-base*, which builds an RF synopsis on the original input dataset. Intuitively, RF-base is similar to applying today's most-recommended synopsis on the input data. Note that RF-base does not consider transformations.
- *RF-shifts*, which builds an RF synopsis on the input after applying all Shift(j, δ) transformers, $1 \leq j \leq n$ and $-\Delta \leq \delta \leq 0$.

The results in Figure 12 demonstrate FPS's superiority over RF-base and RF-shifts in finding a good plan quickly for one-time forecasting queries.

In the technical report, we extend FPS to consider a much larger space of transformers and synopses than Φ. This extension is based on two main observations:

- FPS handles a large space of shift transformations by first applying these transformations to create all corresponding attributes, and then applying efficient attribute traversal and selection techniques to find good transformations quickly. The same technique can be used to consider more transformers, e.g., we consider *log*, *wavelet*, and *difference* in the technical report.

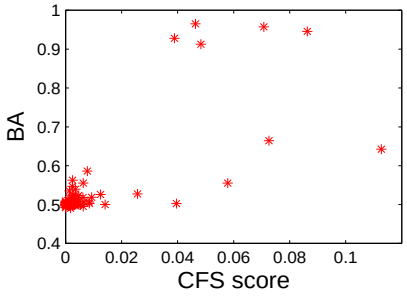

Figure 9: CFS-Score Vs. BA for random subsets(Periodic-large-tb)

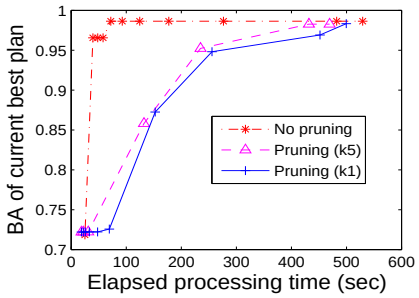

Figure 10: CFS-Score-based pruning (Periodic-large-tb)

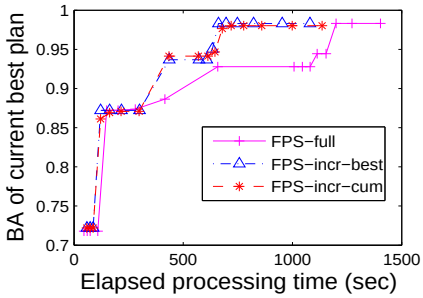

Figure 11: Improvements with sharing (Periodic-large-tb)

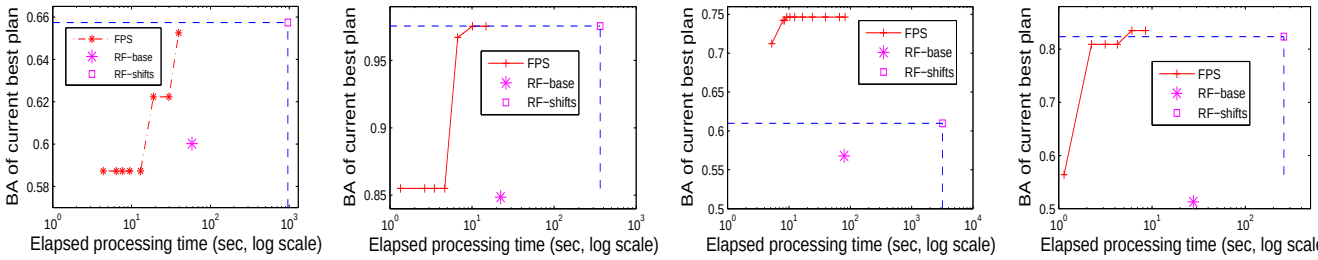

Figure 12: FPS Vs. State-of-the-art synopsis (RF): (a)Aging-real, (b) Complex-syn, (c) Multi-large-tb, (d) Aging-variant-syn

- FPS can first apply simple transformers (e.g., Shift, π) and synopses (e.g., BN) to identify the small subset Ω of original and new attributes that contribute to good plans. Then, more complex transformers and synopses can be applied only on Ω.

8. CONTINUOUS FORECASTING QUERIES

So far we considered one-time forecasting of the form *Forecast*(D, X_i, L) where the dataset D is fixed for the duration of the query. We now extend our techniques to handle continuous queries of the form *Forecast*$(S[W], X_i, L)$ where S is a data stream of relational tuples, and W is a sliding window specification over this stream.

Semantics: The semantics of a *Forecast*$(S[W], X_i, L)$ query is a straightforward extension of the one-time semantics from Section 2. The result of a *Forecast*$(S[W], X_i, L)$ query at time τ is the same as the result of *Forecast*(D, X_i, L), where D is the time-series dataset containing the window W of data in stream S at τ. As time advances, this window of data shifts, and a continuous stream of forecasts will be produced in the result of *Forecast*$(S[W], X_i, L)$.

Example 3. Consider the example continuous query *Forecast*(Usage[Range 5 days],C,1 day) that treats the Usage data in Figure 1(a) as a stream. CQL syntax is used to specify a sliding window containing tuples in the last 5 days. Thus, e.g., on Day 10, a forecast will be output for C for Day 11, based on the data (window) for Days 6-10. □

8.1 FPS-Adaptive (FPS-A)

We could process a *Forecast*$(S[W], X_i, L)$ query by running FPS on the initial window of data to get the best plan p, and then keep producing new forecasts using p as the window of data changes. However, this technique will produce inaccurate forecasts as the predictive patterns in the data change over time. Another option is to rerun FPS from scratch to generate the current best plan whenever the window changes; which is inefficient.

A good algorithm for processing a *Forecast*$(S[W], X_i, L)$ query should use the same plan that FPS would find for each window of data, but incur minimal cost to maintain this plan as the window slides. FPS-A (FPS-Adaptive) is our attempt at such an algorithm. FPS-A exploits the structure (Figure 4) and defaults (Section 7) we established for FPS. FPS-A works as follows:

- The ranked list of all enumerated attributes is maintained efficiently. Because FPS uses LCC for ranking, FPS-A gets two useful properties. First, LCC can be updated incrementally, and batched updates make this overhead very low. Second, recent work [23] shows how tens of thousands of LCCs can be maintained in real-time; we haven't used this work yet since batched updates give us the desired performance.

- If there are significant changes to the ranked list, then the overall "plan structure" is updated efficiently (e.g., the best attribute subset for a chunk may have changed, giving a new best plan).

At all points of time, FPS-A maintains a *reference rank list* R^r composed of k chunks $C_1^r \subset C_2^r \subset \cdots \subset C_k^r$, the best plans $p_1^r, \ldots, p_k^r$ for these chunks, and the overall best plan p_{best}^r. The current p_{best}^r is used for producing forecasts and accuracy estimates at any point of time. The reference values are initialized by running FPS on the initial window of tuples. Figure 13 shows how FPS-A maintains these values as the window slides over time. FPS-A uses two user-defined thresholds: α for detecting significant changes in LCC values, and β that determines the batch size for updating LCC values. The notation $Level(Y)^r$ is used to denote the number of the largest chunk to which attribute Y belongs in R^r. That is, $Level(Y)^r = i$ if $Y \in C_i^r$ and $Y \notin C_{i+1}^r$.

After updating LCC scores, FPS-A computes two sets of attributes: *Gainers* (*Losers*) that moved up (down) one or more levels in the ranked list, and whose LCC values changed significantly. If there are no *Gainers* or *Losers*,

Algorithm *Fa's Adaptive Plan Maintenance (FPS-A)*
Input: A β-sized batch of tuples inserted/deleted from the sliding window for a *Forecast*$(S[W], X_i, L)$ query. R^r, $C_1^r, \ldots, C_k^r$, $p_1^r, \ldots, p_k^r$, p_{best}^r are the current reference values;
Output: The reference values will be updated if required;
1. Do batched update of all LCC values to get the new ranked list R and its chunks $C_1, \ldots, C_k$;
2. For attribute Y, let $LCC(Y,Z)^r$ and $LCC(Y,Z)$ be the reference and current LCC between Y and $Z = X_i(\tau + L)$;
3. For attribute Y, $Level(Y)^r = i$ if $Y \in C_i^r$ and $Y \notin C_{i+1}^r$;
4. For attribute Y, $Level(Y) = i$ if $Y \in C_i$ and $Y \notin C_{i+1}$;
// Attributes that gained LCC significantly and moved up levels
5. $Gainers$ = Set of Y such that $Level(Y)^r > Level(Y)$ AND $|LCC(Y,Z)^r - LCC(Y,Z)| > \alpha$);
// Attributes that lost LCC significantly and moved down levels
6. $Losers$ = Set of Y such that $Level(Y)^r < Level(Y)$ AND $|LCC(Y,Z)^r - LCC(Y,Z)| > \alpha$);
7. **IF** ($|Gainers| = 0$ AND $|Losers| = 0$)
8. No changes are required for reference values. Exit;
9. **FOR** (i going from 1 to number of chunks k) {
10. **IF** (there exists a $Y \in Losers$ such that $Y \in p_i^r$) {
11. Regenerate p_i^r using the attribute subset in p_{i-1}^r and the attributes in $C_i - C_{i-1}$, and using BN synopsis;
12. } /* END IF */
13. **ELSE** {
14. $NewAttrs$ = Set of attributes $Y \in Gainers$ such that $Y \in C_i$ and $Y \notin p_i^r$;
15. If $|NewAttrs| > 0$, then regenerate p_i^r using $NewAttrs$ and current attribute subset in p_i^r, and BN synopsis;
16. } /* END ELSE */
17. } /* END FOR */
18. Update the reference rank list, chunks, and best plan;

Figure 13: FPS-Adaptive (FPS-A)

then the current reference values are fine as is. If one or more attributes in the attribute subset of plan p_i^r for the ith chunk are in *Losers*, then p_i^r may have become suboptimal. So, FPS-A regenerates p_i^r using the best attribute subset for the previous chunk and the attributes at this level (recall Section 5.6). Similarly, if new attributes have moved into the ith chunk, then p_i^r can be updated efficiently to (possibly) include these attributes. FPS-A builds BN synopses, and updates the synopsis in the current best plan after every batch of tuples are processed. While these synopses can be updated incrementally, we have got equally efficient performance from simply rebuilding the synopsis on the few (< 10) attributes chosen by attribute selection.

Notice that FPS-A reacts to changes in the ranked list only when attributes transition across levels. (The threshold α filters out spurious transitions at chunk boundaries.) This feature makes FPS-A react aggressively to changes at the head of the ranked list—where attributes have a higher chance of contributing to the best plan—and be lukewarm to changes in the tail. FPS-A gets this feature from the fact that successive overlapping chunk sizes in FPS increase in a geometric progression.

We now report experiments where we evaluate FPS-A on the three metrics for adaptive query processing defined in [3]: (i) speed of adaptivity (how quickly can FPS-A adapt to changes in stream properties?), (ii) convergence properties (when stream properties stabilize, can FPS-A produce the accuracy that FPS gives for the stable properties?), and (iii) run-time overhead (how much extra overhead does FPS-A incur in settings where stream properties are stable, and in settings where properties change over time?).

We consider continuous queries with a 200-minute window on the input stream. $\alpha = 0.1$ for FPS-A. Figures 14(a) and (b) show FPS-A's speed of adaptivity and convergence properties for two input streams. The input stream used in Figure 14(a) concatenates the Multi-large-tb and Aging-fixed-tb datasets from Table 2 to create a scenario where predictive patterns change because of a change in workload on the monitored system. The second stream is more adversarial where we randomly permute the attributes in the Multi-small-tb dataset over time.

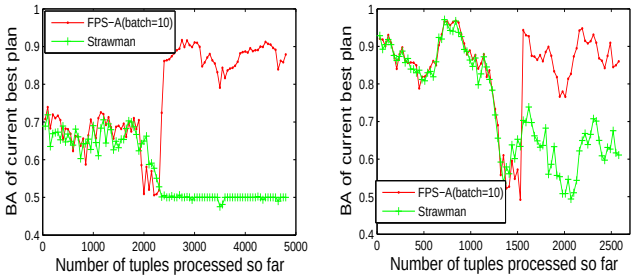

Figure 14: Adaptivity and convergence properties

β	Without property changes			With property changes		
	FPS-A	Strawman	%	FPS-A	Strawman	%
5	32.02	32.83	2.53	33.41	36.69	9.82
10	61.17	62.24	1.74	66.37	60.73	-8.5
15	89.5	90.99	1.56	81.87	88.33	7.89
20	116.27	118.17	1.64	95.50	112.82	18.13
25	129.34	132.10	2.13	108.54	126.24	16.30

Table 4: Run-time overhead. Values for FPS-A and Strawman are tuples processed per second. % is FPS-A's degradation relative to Strawman

We compare FPS-A's performance with that of *Strawman* which runs FPS on the initial window of data to choose the initial best plan. Like FPS-A, Strawman updates the BN synopsis in the current best plan after every batch of tuples have been processed. However, unlike FPS-A, Strawman never updates the set of attributes selected. The X-axis in Figure 14 shows the number of tuples processed so far, and the Y axis shows the estimated accuracy of the current best plan. Note that FPS-A adapts fairly quickly in Figure 14(a) when the stream properties change at $X = 2000$, but Strawman does not because the set of predictive attributes has changed. Comparing the accuracies before and after the change in Figure 14(a) with the respective individual best accuracies for the Multi-large-tb and Aging-fixed-tb datasets in Table 3, shows that FPS-A indeed finds plans comparable to FPS. The behavior in Figure 14(b) is similar.

Table 4 shows the extra overhead for FPS-A over Strawman for the adversarial stream. When there are no changes in stream properties, FPS-A's extra overhead is limited to LCC maintenance; which is around 2%. Even when stream properties change and FPS-A has to update attribute subsets, the worst-case overhead is $< 20\%$. Note that FPS-A's overhead can be lower than that of Strawman if FPS-A finds a smaller attribute subset.

9. RELATED WORK

Time-series forecasting has been a topic of extensive research for years because of its use in many domains like weather and econometric forecasting. Reference [5] is a recent and comprehensive review of this research over the past 25 years. The motivation of making systems and DBMSs easier to manage (ideally self-tuning) has attracted recent interest in forecasting (e.g., [13, 14]). Our work differs from previous work on forecasting in two fundamental ways: (i)

we consider declaratively-specified forecasting queries, and develop automated algorithms for choosing plans composed of transformation, prediction, and synopsis-building operators; (ii) our algorithms balance accuracy against the time to generate forecasts.

Our goal in this paper is not to propose new synopses or transformers for forecasting—plenty of such operators are listed in [5]—but to develop algorithms that can compose existing operators into a good plan for a given query and dataset. While [13, 14] show how various synopses can forecast performance problems and failures in real enterprise environments, transformations are selected manually in [13, 14]. Muscles [21] uses MLR synopses to process continuous forecasting queries over data streams. While Muscles updates synopses incrementally as new tuples arrive, the set of transformations is not updated.

The design of FPS has been influenced by recent comparison studies on synopses and transformations done by the machine-learning community [4, 10, 19]. Reference [4] reports a large-scale empirical comparison of the accuracy (processing time is not considered) of ten synopses, including regression, BN, CART, SVM, and RF. This study found RF to be one of the best synopses available. Reference [19] is a similar study, but with a much smaller scope—only BN and CART synopses are considered. However, [19] evaluates processing time and considers some attribute-selection algorithms. Reference [10] is a comprehensive evaluation of attribute-selection algorithms, with findings similar to ours. None of this work considers algorithms for determining a good combination of synopses and transformations.

Synopses have found many uses recently in database and data stream systems, e.g., in approximate query answering, query optimization, self-tuning, and acquisitional query processing [6, 12, 17]. While most of this work focuses on conventional SQL or XML queries, Fa can benefit from some of it. For example, [23] develops techniques to maintain a large number of LCCs in real-time. Reference [5] urges the time-series forecasting community to develop multivariate synopses that are robust and easy to use as well as good at multi-step-ahead forecasting. Commercial DBMSs have made rapid strides towards becoming self-tuning (e.g., see [17]). However, these systems take a predominantly reactive approach to tuning and diagnosis, which can be made proactive with Fa's accurate and automated forecasting.

Fa can leverage as well as extend work on integrating data-mining algorithms (e.g., [12, 15]) and synopses (e.g., [7]) into DBMSs. *MauveDB* [7] proposes a declarative language and efficient materialization and update strategies for synopsis-based views. None of this work addresses how to choose the best algorithm or synopsis for a specific query or mining task automatically; where ideas from Fa apply.

10. CONCLUSION AND FUTURE WORK

In conclusion, the Fa system enables users and applications to pose declarative forecasting queries—both one-time and continuous—and get forecasts in real-time along with accuracy estimates. We described the FPS algorithm to process one-time forecasting queries using plans composed of operators for transforming data, building statistical models from data, and doing inference using these models. We also described the FPS-A algorithm that adapts plans for continuous forecasting queries based on the time-varying properties of input data streams. Finally, we demonstrated the effectiveness and scalability of both algorithms empirically.

11. REFERENCES

[1] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.

[2] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup E-Commerce Site. Technical report, HPL-1999-62, 1999.

[3] S. Babu. *Adaptive Query Processing for Data Stream Management Systems*. PhD thesis, Stanford University, Sept. 2005.

[4] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Intl. Conf. on Machine Learning*, June 2006.

[5] 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.

[6] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Intl. Conf. on Very Large Databases*, Sept. 2004.

[7] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *ACM SIGMOD Intl. Conf. on Management of Data*, June 2006.

[8] S. Duan and S. Babu. Processing forecasting queries. Technical report, Duke University, Mar. 2007.

[9] Amazon's EC2. `aws.amazon.com/ec2`.

[10] M. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Trans. on Knowledge and Data Engineering*, 15(3), Nov. 2003.

[11] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *ACM SIGMOD Intl. Conf. on Management of Data*, May 1997.

[12] Data Mining Algorithms in SQL Server 2005. `msdn2.microsoft.com/en-us/library/ms175595.aspx`.

[13] R. Powers, M. Goldszmidt, and I. Cohen. Short term performance forecasting in enterprise systems. In *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, Aug. 2005.

[14] R. K. Sahoo et al. Critical event prediction for proactive management in large-scale computer clusters. In *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, Aug. 2003.

[15] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. *Data Mining and Knowledge Discovery*, 4(2-3), 2000.

[16] P. Selinger et al. Access path selection in a relational database management system. In *ACM SIGMOD Intl. Conf. on Management of Data*, June 1979.

[17] Special issue on self-managing database systems. *IEEE Data Engineering Bulletin*, 29(3), Sept. 2006.

[18] K. Vaidyanathan and K. S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, 2(2), 2005.

[19] N. Williams et al. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *Computer Communication Review*, 36(5), 2006.

[20] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, June 2005.

[21] B. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *Intl. Conf. on Data Engineering*, Mar. 2000.

[22] L. Yu and H. Liu. Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. In *Intl. Conf. on Data Engineering*, Aug. 2003.

[23] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Intl. Conf. on Very Large Data Bases*, Sept. 2002.