

# Condition Evaluation for Speculative Systems: a Streaming Time Series Case

X. Sean Wang  
CS Dept., Univ. of Vermont  
xywang@cs.uvm.edu

Like Gao  
CS Dept., Univ. of Vermont  
lgao@cs.uvm.edu

Min Wang  
IBM T.J. Watson  
min@us.ibm.com

## Abstract

Application systems often need to react with certain actions whenever some preset conditions are satisfied. In many cases, the evaluation of these conditions takes long time, but some prediction of the results can be obtained rather quickly. In this situation, speculation may be a good idea. That is, the system takes predictions (speculation) to prepare (such as prefetch) for the possible reaction. Obviously, the risk is wasted efforts due to false alarms. Higher precision prediction results in less waste, but takes longer time and may reduce/eliminate the opportunity for speculation. A balance needs to be struck. A quality-driven prediction subsystem is thus necessary, so that the “user” of the prediction subsystem can impose quality (in terms of precision and response-time) requirements.

This paper focuses on such a prediction subsystem with conditions on streaming time series. Two problems need to be solved: how to predict the precision and how to achieve the required precision in an optimized way. The paper introduces a prediction model to tackle the first problem, and presents an algorithm to attack the second. Experiments show that the prediction subsystem works well.

## 1 Introduction

A speculative system is one that uses some kind of risk taking mechanism to achieve an overall gain. Prefetching in various scenarios, such as in processors, operating systems, and database systems, is a typical example. To speculate is generally to carry out some

activity based on the prediction of the forthcoming requests, thereby saving some time when the actual requests are processed. The risk of speculation is wasted efforts while the gain is the potential overall increase in throughput.

Obviously, a number of factors affect the performance of a speculative system. Prediction precision is an important one. Usually, the higher the prediction precision is, the more the system gains. However, in certain situations, higher prediction precision takes longer time to produce. An overall strategy is needed that trades off response time with precision.

In this paper, we study a condition-driven system that takes reactive measures (by a reactive subsystem) whenever some preset conditions are satisfied (evaluated by a parallel condition evaluation subsystem). We assume the conditions evaluation subsystem contains a prediction subsystem that is responsible to provide a prediction to the truth values of the conditions. We focus on this prediction subsystem.

Since precision and response time of the prediction (by the prediction subsystem) are usually positively correlated, a balance must be struck between them in order to achieve overall gain. It is thus imperative for the prediction subsystem to have the ability to trade time with precision. In other words, we require that the prediction subsystem be able to satisfy some QoS (quality of service) requirements.

More specifically, given a set of conditions, the prediction subsystem will partition them into two sets, one true-set and one false-set. We measure the precision of the two sets by false positive/negative ratios. The smaller these values are, the more precise the prediction is. Another view is that the predictions are approximate answer to the evaluation of the conditions.\* The response time is the time when the last condition in the true-set is produced by the prediction subsystem. This is due to the fact that most systems respond to conditions that become true (otherwise, it is usually not difficult to negate conditions).

---

Copyright held by the author(s).

Proceedings of the Second Workshop on Spatio-Temporal Database Management (STDBM'04), Toronto, Canada, August 30th, 2004.

---

\*Hence, we will use *prediction* and *evaluation* interchangeably when no confusion arises.

In order to implement a quality-driven prediction subsystem, it is important to have the ability to measure the (intermediate) result quality during the evaluation process. While it is straightforward to measure the response time, it is impossible to measure the accuracy (i.e., false positive and negative ratios) precisely when an approximation method is used. Indeed, the precise accuracy can only be measured *a posteriori*, i.e., only after we know the actual evaluation results of the conditions. So we advocate measuring accuracy in an *a priori* manner, i.e., the false ratios are estimated based on prior knowledge. To do this, we build a prediction model based on historical data analysis. At the evaluation time, we use this prediction model to derive precision estimates. When the precision based on the prediction model is not enough to satisfy the “user” requirements, the prediction subsystem needs some processing to increase the precision.

Since the evaluation procedure is based on the prediction model that is probabilistic in nature, our system cannot guarantee the satisfaction of accuracy in a strict sense. Instead, similar to the *soft* quality of service (QoS) concept in computer network [9], the system satisfies the accuracy requirements in a “soft” manner. That is, the system guarantees with enough confidence that the *expected* accuracy will not exceed given thresholds. Specifically, the system allows the “user” (i.e., the reactive subsystem) to impose the following quality constraints:

- *Response time constraint:* All evaluation results must be reported within a given time limit.
- *Accuracy constraints:* The expected false positive and negative ratios must not exceed the given thresholds, with enough confidence (the confidence is deduced from the prediction model).

Since the prediction subsystem may not be able to satisfy both constraints simultaneously, especially when the constraints are strict and system resource is limited. We choose to let the user (i.e., the reactive subsystem) impose a precision requirement, and let the prediction subsystem optimize on the response time. In other cases, the user may want to impose a response time requirement (and then decide what to do based on the prediction precision), but this is beyond the scope of this workshop paper.

The rest of the paper is organized as follows. In Section 2, we briefly discuss an application where a speculative system with a condition prediction subsystem can be useful. In Section 3, we give some formally definitions of our problem, and in Section 4, we introduce our prediction model. We outline our evaluation algorithm in Section 5 and present experimental results in Section 6. We review related work in Section 7, and conclude the paper with Section 8.

## 2 Speculating Situation Manager

In [3], active systems are studied with an underlying situation manager, where a situation is a reactive entity that receives events as an input, combines composition filtering with content filtering, and detects situations as an output [3]. Many applications require reactions to situations (rather than single events). The high-level architecture of an application that uses the situation manager can be like that in Figure 1 [3]. An example situation can be that “within the same day for at least 5 times, a customer sells after buys the same stock” [3].

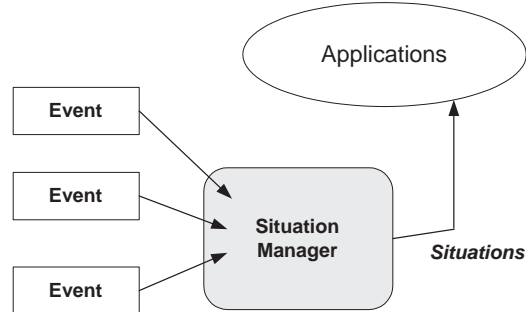


Figure 1: Situation manager high-level architecture.

A situation can take a long time to emerge from the situation manager due to two possible reasons. Firstly, the evaluation of conditions that involve complex operations may take a long time after all the events have arrived.<sup>†</sup> For some complex operators, some kind of approximation can be used to produce a prediction for the conditions. Secondly, the temporal distance between the first and the last events for the situation can be great. In the above example, only when the 5<sup>th</sup> time that the customer sells after buys the same stock, the situation appears. This gives room for a speculative application system. Indeed, in the above example, when the 5<sup>th</sup> buy of a stock by the same customer occurs, we probably have enough confidence to speculate that the 5<sup>th</sup> sell will occur. In this case, we can tell the application with some confidence that the situation will occur so that the application can start to prepare reactive measures for the situation. If the situation does occur, the application can react faster. If the situation in the end does not occur, then the application just wasted some efforts (and rollback may be necessary). Depending on the application, this may be a risk worth taking.

In other words, when situations take long time to emerge and can be predicted with some confidence, it may be advantageous to predict certain situations. In this case, the situation manager becomes a speculating one, and applications become speculative. See Figure 2. In this case, the speculative application should

<sup>†</sup>AMIT does not allow complex operators like nearest neighbor search.

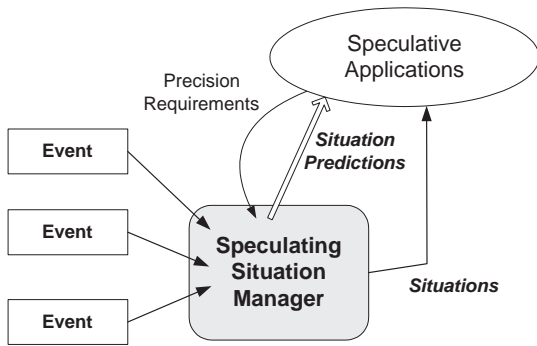


Figure 2: Speculating situation manager.

tell the situation manager how precise the prediction should be in order to achieve the best overall gain.

The question we explore in this paper is how to provide predictions to the true-set and false-set for the given conditions. Obviously, given enough time, the predictions can be made as precise as possible (the “worst” case is to wait until the conditions are evaluated fully, and “predictions” become actual values). The question is how to respond in the fastest way as long as the precision requirement is satisfied. In this paper, we assume the conditions are on streaming time series as defined below.

### 3 Basic Definitions

#### Streaming time series

A *time series* is a finite sequence of real numbers and the number of values in a time series is its *length*. A *streaming time series*, denoted  $s$ , is an infinite sequence of real numbers. At each time position  $i$ , however, the streaming time series takes the form of a finite sequence, assuming the last real number is the one that arrived at time  $i$ .

#### Condition on streaming time series

In general, a condition can be any user-defined predicate on streaming time series, and needs to be evaluated after every data arrival.

As an example, given (finite) time series  $x$  and  $y$  of the same length  $l$ , we may define the following correlation coefficient function:

$$\text{corr}(x, y) = \frac{\sum_{i=1}^l (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^l (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^l (y_i - \bar{y})^2}},$$

where  $\bar{x}$  and  $\bar{y}$  are the mean values of  $x$  and  $y$ , respectively. Correlation coefficient quantitatively measures the degree of correlation between two time series in terms of how closely they are related to each other.

When dealing with streaming time series  $s^1$  and  $s^2$ , we only look at their sub-series within a sliding window of size  $w$ . That is, for each time  $t$  ( $t \geq w$ ), we define the function values of  $s^1$  and  $s^2$  at time  $t$  by using

$x = s^1[t - w + 1, \dots, t]$  and  $y = s^2[t - w + 1, \dots, t]$ . Correspondingly, we use  $\text{corr}(s^1, s^2, w)$  to denote the function with sliding window size of  $w$  on streaming time series  $s^1$  and  $s^2$ . The function yields a value at each given time  $t$  ( $t \geq w$ ). Using this function, we can naturally form conditions on streaming time series, which are commonly used to monitor interesting trend of streaming time series [25].

- $|\text{corr}(s^1, s^2, w)| > 0.75$ : the (absolute) correlation between two streaming time series is high;
- $|\text{corr}(s^1, s^2, w_1)| > 2|\text{corr}(s^1, s^3, w_2)|$ : the (absolute) correlation between  $s^1$  and  $s^2$  is two times greater than that between  $s^1$  and  $s^3$ .

Note that the above conditions are quite trivial in terms of computation, but will be used in this paper for illustrative purposes. In real application, conditions can be quite complex, and can sometime involve accesses to large databases. For example, a condition may specify that one of the near neighbors (among a large collection of time series objects) of the streaming time series shows a particular property.

#### Quality measures

We denote a set of conditions as  $C = \{c_1, c_2, \dots, c_n\}$  and the reported evaluation result of condition  $c_i$  at time position  $t$  as  $r(c_i, t)$ . We denote the precise evaluation result (the actual value of the given condition when a precise evaluation process is used) of  $c_i$  at time position  $t$  as  $R(c_i, t)$ . Obviously, we have  $r(c_i, t) \in \{\text{True}, \text{False}\}$ , and  $R(c_i, t) \in \{\text{True}, \text{False}\}$ .<sup>‡</sup> Note that  $r(c_i, t)$  may not be equal to  $R(c_i, t)$  due to the approximate nature of the system. Let  $C_T$  denote all the conditions in  $C$  whose reported results are True at time position  $t$ , i.e.,  $C_T = \{c_i | c_i \in C \text{ and } r(c_i, t) = \text{True}\}$ . Similarly, let  $C_F = \{c_i | c_i \in C \text{ and } r(c_i, t) = \text{False}\}$ . We call  $C_T$  and  $C_F$  the reported-True and reported-False sets, respectively. The two sets are disjoint and  $C = C_T \cup C_F$ .

Using the above notation, we define the following three parameters to measure the quality of an evaluation system at each time position  $t$ , with a smaller value meaning better quality.

1. *Response Time, RT*, is the duration from the data arrival time  $t$  to the time when last condition  $c_i$ ,  $r(c_i, t) = \text{True}$ , is reported.
2. *False Positive Ratio, FPR*, of a reported-True set  $C_T$  is the fraction of the conditions (among all the conditions in  $C_T$ ) whose actual values are False. We define  $FPR = 0$  if  $C_T$  is an empty set.

<sup>‡</sup>In the rest of the paper, we may omit  $t$  and use  $r(c_i)$  ( $R(c_i)$ ) to denote the reported evaluation result (actual value) of condition  $c_i$  at time position  $t$  when the context is clear.

3. *False Negative Ratio, FNR, of a reported-False set*  $C_F$  is the fraction of the conditions (among all conditions in  $C_F$ ) whose actual values are True. We define  $FNR = 0$  if  $C_F$  is an empty set.

Note that response time is defined only on conditions that are reported true. This is because the particular situation we are dealing with in which the situation manager only needs to enact actions when corresponding conditions become true, and does nothing in other cases. Other situations may call for different definitions, such as averaged elapsed time over all reported true conditions, etc.

## 4 Prediction Model

While it is easy and straightforward to measure the quality parameter  $RT$ , it is difficult to measure the other two quality parameters,  $FPR$  and  $FNR$ . For example, according to the definition of  $FPR$ , we need to know the actual values of all conditions in a reported-True set  $C_T$  when to calculate  $FPR$ . Of course, this is an unrealistic requirement since the underlying assumption is that we do not know the actual values of all the conditions. A practical alternative is to build a prediction model using historical evaluation results and calculate the *expected*  $FPR$  and  $FNR$  based on the model in a probabilistic manner.

### Probability distribution of an evaluation result

For each condition  $c_i$ , we define a random variable  $X_i$  to state the outcome of its evaluation. Clearly,  $X_i$  follows Bernoulli distribution  $X_i \sim B(\rho_i)$ , that is,

$$X_i = \begin{cases} 1 & \text{if } R(c_i) = \text{True} \\ 0 & \text{if } R(c_i) = \text{False} \end{cases} \quad \text{with} \quad \begin{cases} P(X_i = 1) = \rho_i \\ P(X_i = 0) = 1 - \rho_i \end{cases}$$

where the mean  $\rho_i$  is also called the expected value of  $X_i$ . Note in this paper, we make the simplifying assumption that all  $X_i$ 's are mutually independent.

Depending on how the system treats  $c_i$ , we see three different cases for  $\rho_i$ : (1)  $c_i$  is precisely evaluated. In this case, we have  $\rho_i = 1$  if  $c_i$  is evaluated to be True and  $\rho_i = 0$  if  $c_i$  is evaluated to be False. (2)  $c_i$ 's result is reported based on an approximation procedure, e.g., prediction. In this case,  $\rho_i$  cannot be known exactly. Instead, its estimate  $\hat{\rho}_i$  will be used. This  $\hat{\rho}_i$  is a random variable following certain distribution, due to the fact that the approximation procedure gives the estimate with some confidence. (3) Other cases where  $\rho_i$  is unknown. We now discuss how to model the distribution of  $\rho_i$  for the latter two cases.

In Case (2), assume  $c_i$  is estimated by an approximation procedure that is based on  $N$  historical evaluation results (samples). Then the estimate of  $\rho_i$ ,  $\hat{\rho}_i$ , can be approximated by a normal distribution function  $Norm(\mu_i, \sigma_i^2)$ , where the mean value  $\mu_i = \bar{X}_i$  and the variance  $\sigma_i^2 = \frac{(1-\mu_i)\mu_i}{N}$ , i.e.,  $\hat{\rho}_i \sim Norm(\mu_i, \frac{(1-\mu_i)\mu_i}{N})$ . (Here,  $\bar{X}_i$  denotes the sample mean.)

For Case (3), since  $\rho_i$  is unknown, its estimate  $\hat{\rho}_i$  is used again. However, different from Case (2), the distribution of  $\hat{\rho}_i$  is modeled as  $Norm(0.5, 0.25)$ . Mean value 0.5 implies that the chances of  $c_i$  being true or false are the same. The variance value 0.25 is the maximum variance that an estimate of  $\rho$  can have.

For all the three cases, the estimate of  $\rho_i$  can be viewed as a random variable that follows normal distribution. (Case (1) is a special case of normal distribution.)

### Example prediction model

To get  $\hat{\rho}_i$  when condition  $c_i$  is predicted, we propose to build a prediction model by analyzing the historical evaluation results for each condition. We adopt the data mining approach in [14] and histogram techniques in [20] to build the prediction model. Without loss of generality, we assume each atomic condition is in the form of  $f() > \gamma$ , where  $f$  is an arithmetic expression with the two basic functions and  $\gamma$  is a constant. We partition the range of  $f$  (values of  $f$ ) into buckets. Each bucket is denoted by its boundaries  $(v_1, v_2]$ .

We build an *i-step look-ahead* prediction model based on historical data, which are the precise evaluation results in a long run. At each time position, function  $f$  is evaluated precisely. For a function  $f$ , a bucket  $(v_1, v_2]$ , and a fixed look-ahead step  $i$ , we obtain a count as follows: find all the time positions in the long run when the value of  $f$  falls into the bucket  $(v_1, v_2]$ . Randomly pick  $N$  of these positions, say,  $t_1, \dots, t_N$ . Then check the condition  $f() > \gamma$  at time positions  $t_j + i$  for  $j = 1, \dots, N$ , and record the percentage of times the condition is true. Hence, we can construct a histogram for each  $f$  and  $i$ : the buckets correspond to a partition of the range of  $f$ , and each bucket is associated with a percentage value which is obtained in the way discussed above. This histogram can be refined via histogram building techniques (e.g., splitting and merging) [2].

The histogram can be represented by a curve (interpolated) instead of bars for clarity: the  $x$ -axis represents the buckets (or the range of  $f$ ), and the  $y$ -axis represents the percentages (which we call  $\mu$ ). Fig. 3 shows six example histograms for the condition  $|corr(s^1, s^2, 50)| > 0.75$ . (In these examples, the history has 20,000 time positions and we take 50 samples for each bucket in the histogram.) As an example, assume at time  $t$ , we obtain  $|corr(s^1, s^2, 50)| = 0.85$  through a precise evaluation procedure. If we choose to look ahead 5 steps, we look up the corresponding curve in Fig. 3 and obtain  $\mu = 0.8$  for  $f = 0.85$ . This means if the correlation value is 0.85 (or  $-0.85$ ) at time  $t$ , we can predict that this condition (absolute correlation value is greater than 0.75) is true at time  $t + 5$  with a probability of 0.8.

The above only gives an example on how a prediction model might be obtained. Different kind of con-

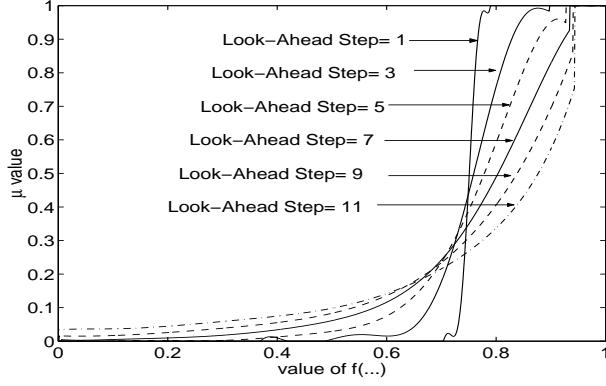


Figure 3: A sample prediction model.

ditions call for different models, and different ways to build the models.

In general, a prediction model for each condition is built based on some feature values (like in the above example). We assume that when a condition is precisely evaluated, the corresponding feature values (used for prediction) are extracted. When the prediction of a condition is required, we will look back in time to find the nearest time position when the condition was precisely evaluated. We use then-extracted feature values and the prediction model to predict the probability for the condition to be true.

### FPR and FNR constraints

With the prediction model, given a reported-True set of size  $m$ , we can derive its expected false positive ratio, denoted  $E(FPR)$ . Indeed, we can prove

$$E(FPR) \sim Norm(\mu, \sigma^2)$$

where  $\mu = \sum_{i=1}^m (1 - \mu_i) / m$  and  $\sigma^2 = \sum_{i=1}^m \sigma_i^2 / m^2$ . Here  $\mu_i$  and  $\sigma_i$  take values from the corresponding cases.

Similarly, we can prove that the expected *FNR* of a reported-False set of size  $m$  is also a normally distributed random variable with  $\mu = \sum_{i=1}^m \mu_i / m$  and  $\sigma^2 = \sum_{i=1}^m \sigma_i^2 / m^2$ .

We are now ready to define false positive ratio (*FPR*) constraint and false negative ratio (*FNR*) constraint by using the expected *FPR* and *FNR*.

**Definition** An *FPR*-constraint is in the form of a pair  $\tau_{FPR} = (\theta_E, \alpha)$  ( $0 \leq \theta_E, \alpha \leq 1$ ). A set of reported-True conditions  $C_T$  satisfies  $\tau_{FPR}$  if  $P\{E(FPR) \leq \theta_E\} \geq \alpha$ . We call  $\theta_E$  and  $\alpha$  the *expected-mean threshold* and the *confidence threshold*, respectively.

Intuitively, the smaller the  $\theta_E$  value and the greater the  $\alpha$  value, the “tighter” an *FPR*-constraint  $\tau_{FPR}$  is. Formally, we define a partial order  $\tau'_{FPR} \leq \tau''_{FPR}$  if  $\theta'_E \leq \theta''_E$  and  $\alpha' \geq \alpha''$ , and we say  $\tau'_{FPR}$  is a tighter

*FPR*-constraint than  $\tau''_{FPR}$ . Among all such *FPR*-constraints that a reported-True set  $C_T$  satisfies, we call the “tightest” one as the *FPR-quality* of set  $C_T$ . For simplicity, we fix the value of  $\alpha$  in all constraints, and thus, the one having the smallest  $\theta_E$  will always give the highest *FPR-quality*.

Symmetrically, we can define *FNR*-constraint  $\tau_{FNR}$  and *FNR-quality* of a reported-false set  $C_F$ .

## 5 Evaluation algorithm

As mentioned earlier, in our speculative situation manager, an important decision is how precise (in terms of false-positive/negative ratios) the underlying condition evaluation system is. This decision is based on how much saving the approximated algorithm can save (in terms of response time). A basic problem for the underlying condition evaluation system is the following optimization problem: use the minimum amount of time to achieve the required precision.

More precisely, given accuracy constraints for both  $C_T$  and  $C_F$  (i.e.,  $\tau_{FPR}$  and  $\tau_{FNR}$ ), we need to minimize the response time. We use a greedy algorithm for this optimization problem, as shown in Fig. 4. The basic idea is to increase the size of  $C_T$  and  $C_F$  aggressively, and at the same time, try to report as early as possible those conditions in  $C_T$ .

In the algorithm, we assume that if a condition is reported true, then its  $\mu$  value must be greater than threshold 0.5. That is, we don’t want to risk it if the chance of condition to be true is small. Likewise, if a condition is reported to be false, its  $\mu$  value must be less than (or equal to) 0.5. Different system may require other threshold values other than 0.5.

The algorithm starts with using `InitExpandCT` to get initial report-True set  $C_T$ . The procedure `InitExpandCT` is to take all the conditions with the highest  $\mu$  values (no less than 0.5) as long as the *FPR*-constraint is satisfied. The conditions in  $C_T$  are reported to be True. These are the conditions that can be reported True without doing any precise evaluation. This is Step 1.

After Step 1, we need to precisely evaluate conditions in order to report them true (without violating either  $\tau_{FPR}$  or  $\mu > 0.5$ ). In Step 2, as a greedy algorithm, we pick up the condition having the next highest  $\mu$  value. This is the one immediately after the conditions in the initial  $C_T$  in the  $\mu$ List. Hence, we pick it (i.e.,  $c_{i_T}$ ) up to precisely evaluate. If the condition is evaluated True, we add it to  $C_T$  and try to expand  $C_T$  with no precise evaluation again (by calling a Procedure `ExpandCT`, which basically tries to grab the conditions with the next highest  $\mu$  values as long as the *FPR*-constraint is satisfied), report the conditions in the expended  $C_T$  and keep going. If the condition is evaluated False, then we just keep going to precisely evaluate the next condition, since we are still not able to expand  $C_T$  without a condition evaluated true.

Consts.:	$\tau_{FPR}$ and $\tau_{FNR}$ constraints
Goal:	minimize response time ( $RT$ )
Step 1.	Form and report the reported-True set: Initialize $z_T = 0$ and $z_F = 0$ ; $[z_T, i_T] = \text{InitExpandC}_T(z_T)$ ; Report $c_1, \dots, c_{i_T-1}$ as True;
Step 2.	Process all conditions $c_i$ with ( $\mu_i > 0.5$ ): Do loop until ( $\mu_{i_T} \leq 0.5$ ) or ( $i_T > n$ ): { - Precisely evaluate $c_{i_T}$ . Two outcomes: - If $c_{i_T}$ is evaluated False, update $z_F$ with an extra reported-False condition, continue the loop with $i_T = i_T + 1$ ; - If $c_{i_T}$ is evaluated True, then - Update $z_T$ w/ an extra rpt.-True cond.; - $[z_T, \Delta_T] = \text{ExpandC}_T(z_T, i_T + 1)$ - Report $c_{i_T}, \dots, c_{i_T+\Delta_T}$ as True; - Update $i_T = i_T + \Delta_T + 1$ ; }
Step 3.	Form the reported-False set: $[z_F, i_F] = \text{InitExpandC}_F(z_F)$
Step 4.	Process all conditions $c_{i_T}$ with ( $\mu_{i_T} \leq 0.5$ ): Continue to use the same $i_T$ from Step 2, do loop until ( $i_T > i_F$ ). { - Precisely evaluate $c_{i_T}$ . Two outcomes: - If $c_{i_T}$ is evaluated True, report $c_{i_T}$ as True and continue the loop with $i_T = i_T + 1$ ; - If $c_{i_T}$ is evaluated False, then o update $z_F$ w/ an extra rpt.-False cond.; o $[z_F, \Delta_F] = \text{ExpandC}_F(z_F, i_F)$ ; - update $i_F = i_F - \Delta_F$ and $i_T = i_T + 1$ ; }
Step 5.	Report as False all those conditions that were not reported True.

Figure 4: Algorithm `QualEval`.

During Step 2, if we run out of conditions in  $\mu\text{List}$ , we can stop (just report all the conditions that were evaluated False as false and thus achieve  $FNR = 0$ ). If the  $\mu\text{List}$  is not exhausted, then we need to reach the first condition in the  $\mu\text{List}$  such that its  $\mu$  value is no greater than 0.5.

Once we only have conditions with  $\mu$  no greater than 0.5, we need to precisely evaluate them and report them as soon as they are evaluated True. However, there is a chance we may be able to report them False. Therefore, Step 3 tries to get the maximum set of conditions to report False without precise evaluation (note that all the conditions that were evaluated False need to be taken into account, hence the  $z_F$  value may not start with 0 in Step 3). The procedures `InitExpandCF` and `ExpandCF` are analogous to `InitExpandCT` and `ExpandCT`, respectively.

After Step 3, if we still have conditions that need to be processed (i.e., if  $i_T \leq i_F$ ), we will pick them up to evaluate. Since we want to minimize the response time for the conditions in  $C_T$ , we precisely evaluate the

conditions starting from those with greater  $\mu$  values. Again, if any condition is evaluated False, we will try to expand  $C_F$ .

It is easily seen that the algorithm is correct since both  $FPR$  and  $FNR$ -constraints are both satisfied.

## 6 Experimental Results

In this section, we present our experimental results, showing that the algorithm effectively achieves its optimization goal and satisfies the quality requirements.

We first describe the data set, condition set, and performance parameters we used in our experiments.

**Data set:** We generate synthetic data for experiments. The data set consists of 100 streaming time series. Each time series is independently generated with a random walk function. For stream  $s$ ,  $s_i = s_{i-1} + \text{rand}$ , where  $\text{rand}$  is a random variable uniformly distributed in the range of  $[-0.5, 0.5]$ .

**Condition set:** Our condition set includes 400 conditions defined over these 100 streams. Each condition may contain one or more correlation functions. Each correlation function is defined on two streams that are picked up randomly from the 100 streams, with its sliding window size being randomly chosen from  $[50, 1000]$ .

**Performance parameters:** We use the four quality parameters described in Section 3 (i.e.,  $RT$ ,  $FPR$  and  $FNR$ ) to measure the performance of our algorithms. Note that  $FPR$  and  $FNR$  are all real numbers in  $[0, 1]$  and can be computed precisely by comparing the reported evaluation results with the precise evaluation results (done for the purpose of performance evaluation). The response time is measured by the number of conditions that are precisely evaluated (either to True or False) before all the conditions in the reported-True set are reported. By using this measure (instead of using real time), we can clearly separate the overhead of the optimization procedure and the condition evaluation time.

We now show the performance of `QualEval`. In our experiment, we set the confidence threshold  $\alpha = 95\%$  for both  $FPR$ - and  $FNR$ -constraints. We vary the expected-mean threshold  $\theta_E$  from 0.05 to 0.3 in different runs, and execute the algorithm for 1,000 time positions in each run.

Fig. 5(a) and (b) show the evaluation quality achieved in terms of actual  $FPR$  and  $FNR$ . The two plots of Fig. 5(a) present the actual  $FPR$  and  $FNR$  values at each time position for 200 time positions with  $\theta_E = 0.01$  (for both  $\tau_{FPR}$  and  $\tau_{FNR}$ ). We can see that these actual  $FPR$  ( $FNR$ ) values are in the range  $[0.01, 0.04]$  with a mean of 0.008 (which is very close to the given  $\theta_E = 0.01$ ). Fig. 5(b) presents how well  $FPR$  ( $FNR$ ) constraints with various mean thresholds (varying from 0.01 to 0.3) are satisfied by our algorithm. We calculate the average of the actual  $FPR$  ( $FNR$ ) values over 1000 time positions for each run, and we can see

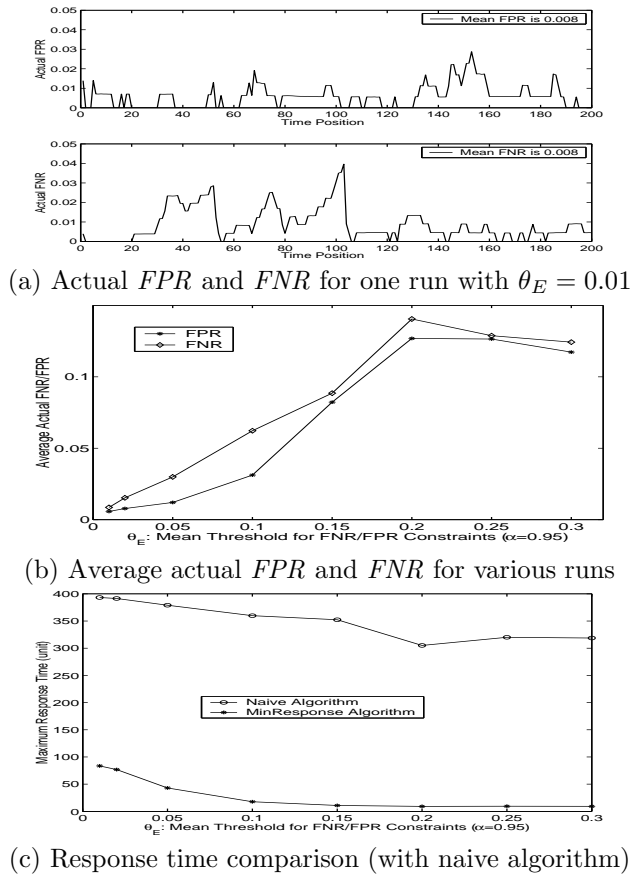


Figure 5: Performance of QualEval.

that the average is either below or very close to the corresponding given expected-mean threshold  $\theta_E$  for all the runs. Note that when the given  $\theta_E$  becomes large enough (i.e., greater than 0.2 in Fig. 5(b)), the actual *FPR* (*FNR*) values tend to become some constants that are determined by the prediction model. The little drop of the curves in this figure is due to experimental variance.

Fig. 5(c) shows the performance of QualEval in terms of response time. For comparison, a naive algorithm is implemented: It randomly picks up a condition to evaluate precisely, until it has reported  $k$  Trues, where  $k$  is the number of real Trues in the reported-True set from QualEval (i.e.,  $k$  is the number of  $c_i$ s such that  $R(c_i) = \text{True}$  and  $c_i \in C_T$ ). This is to make the naive algorithm report the same number of true conditions. We compare the response time of QualEval with this naive algorithm for different runs with  $\theta_E$  values in  $[0.01, 0.3]$ . We can see that QualEval consistently outperforms the naive algorithm. Note that the response time of QualEval decreases as  $\theta_E$  increases, because the greater the  $\theta_E$  value, the coarser approximation is allowed, and thus fewer precise evaluations are needed.

The performance gain of QualEval is significant. For example, given  $\theta_E = 0.01$ , QualEval only takes about 1/15 time of the naive algorithm, but maintains the quality of *FPR* and *FNR* at around 1%. When  $\theta_E$  is set to higher values, the performance gain becomes more significant. Note that the confidence threshold  $\alpha$  is set to 95% in all the experiment reported here. Given the same expected-mean threshold  $\theta_E$  for both *FPR*- and *FNR*-constraints, smaller  $\alpha$  will yield faster response. We omit the experimental results on using various  $\alpha$  settings in this paper.

## 7 Related Work

Speculation has been used in various applications such as processor design, cache implementation, and buffer management in OS. Recently, Polyzotis and Ioannidis [22] described a speculative query processing system, where the system anticipates the user queries by predicting from partially entered query statements. Our design follows the same approach.

The quality-driven aspect of our work is similar to the QoS concept in computer network [12, 19]. QoS in computer network allows the end users to specify their requirements on different quality metrics (e.g., service availability, delay, delay variation, throughput, and packet loss rate). The network system needs to guarantee certain levels of service quality based on these requirements. This paper adapts the QoS concept into condition evaluation on streaming time series and presents a basic design strategy for developing such a quality-driven system.

Aurora [24, 1, 8] seems to be the only data stream processing system that contains a QoS component. In Aurora, a user may register an application with a QoS specification that provides the user's preference on the performance and quality of this application. These QoS specifications serve to drive policies for scheduling and load shedding.

With limited resources, using approximation techniques in processing continuous queries on data streams has been studied in [11, 13, 10, 15, 25]. However, most approximate evaluation strategies, and even some precise evaluation strategies, only consider one quality aspect and neglect the others. For example, Chain [6] minimizes the memory usage without considering the response time at all. Our work differs from all such work in that our strategy takes different user-specified quality requirements into consideration to guide the evaluation procedure. In other words, the satisfaction of quality requirements in our system is provided by run-time dynamic adjustments instead of by static algorithm design. This advantage allows our system to handle different constraints.

In Statstream system [25], Zhu and Shasha presented an efficient approximation method to calculate all pair-wise correlations for a set of streaming time series. In this paper, we use correlation function to

construct conditions. While [25] aims at building an approximate evaluation system that can process correlations efficiently, we focus on satisfying the user's quality requirements.

## 8 Conclusion

In this paper, we studied a condition prediction subsystem that considers user-specified quality requirements. We argued that such a system is necessary for speculative systems. We used statistical analysis to derive the likelihood of a condition to be true at a time position. By using this likelihood and the associated confidence (due to finite sampling), we estimated the quality of our predictions. Based on this prediction method, we designed an algorithm to produce predictions satisfying precision requirements. Our experiments showed that the algorithm is effective.

The prediction subsystem presented in this paper works well with simple correlation conditions and synthetic data sets. To make this subsystem applicable to real applications, it would be interesting to extend the prediction subsystem to handle more general conditions such as those in AMIT, or those involving complex operators such as nearest neighbor searches. As future work, it is also interesting to study how to improve the subsystem when some assumptions made in this paper do not hold anymore (e.g., streaming time series come from some common sources and are not independent). Another interesting research direction is to design algorithms that satisfy other types of quality requirements. For example, a user may want to impose a response time limit while requiring the system to achieve the best precision.

## References

- [1] D. J. Abadi et. al. Aurora: A data stream management system. In *SIGMOD*, 2003.
- [2] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *SIGMOD*, pages 181–192, 1999.
- [3] A. Adi, D. Botzer, and O. Etzion. The Situation Manager Component of Amit - Active Middleware Technology. In the 5<sup>th</sup> International Workshop on Next Generation Information Technologies and Systems (NGITS), pages 158–168, 2002.
- [4] D. Anderson et. al. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES). Tech. Report SRI-CSL-95-06, SRI, 1995.
- [5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. In *Journal of the ACM*, 45(6), pages 891–923, 1998.
- [6] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator scheduling for memory minimization in data stream systems. In *SIGMOD*, pages 253–264, 2003.
- [7] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: a multidimensional workload-aware histogram. In *SIGMOD*, pages 211–222, 2001.
- [8] D. Carney et. al. Monitoring streams - a new class of data management applications. In *Very Large Data Bases*, 2002.
- [9] CISCO. Quality of Service (QoS). On-line. [http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito\\_doc/qos.htm](http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm), 2003.
- [10] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, pages 40–51, 2003.
- [11] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, pages 61–72, 2002.
- [12] P. Ferguson and G. Huston. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley & Sons, 1998.
- [13] S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *SIGMOD*, pages 265–276, 2003.
- [14] L. Gao, M. Wang, X. S. Wang, and S. Padmanabhan. A learning-based approach to estimate statistics of operators in continuous queries: a case study. In *DMKD*, June 2003.
- [15] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, pages 79–88, 2001.
- [16] J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams. In *ICDE*, 2003.
- [17] L. Lim, M. Wang, and J. S. Vitter. SASH: A self-adaptive histogram set for dynamically changing workloads. In *VLDB*, 2003.
- [18] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, pages 101–110, 2000.
- [19] D. McDysan. *QoS and Traffic Management in IP and ATM Networks*. McGraw-Hill Osborne Media, 1999.
- [20] V. Poosala. *Histogram-based Estimation Techniques in Databases*. PhD thesis, University of Wisconsin, 1997.
- [21] P. A. Porras and P. D. Neumann. EMERALD: Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, 1997.
- [22] N. Polyzotis, and Y. Ioannidis. Speculative Query Processing. First Biennial Conference on Innovative Data Systems Research (CIDR), 2003.
- [23] S. Ross. *A First Course in Probability*. Prentice Hall, 2001.
- [24] N. Tatbul, U. etintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.
- [25] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.