

# Quality-Driven Evaluation of Trigger Conditions on Streaming Time Series

(Extended Abstract)

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

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

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

## ABSTRACT

For many applications, it is important to evaluate trigger conditions on time series streams. In a resource constrained environment, users' needs should ultimately decide how the evaluation system balances the competing factors such as evaluation speed, result precision, and load shedding level. This paper presents a basic framework for evaluation algorithms that takes user-specified quality requirements into consideration. Three optimization algorithms, each under a different set of quality requirements, are developed in the framework: (1) minimize the response time given accuracy requirements and without load shedding; (2) minimize the load shedding given a response time limit and accuracy requirements; and (3) minimize one type of accuracy errors given a response time limit and without load shedding. Experiments show that these optimization algorithms effectively achieve their optimization goals while satisfying the corresponding user-specified quality requirements.

## 1. INTRODUCTION

There are many applications in which it is important to monitor the behavior of some processes and to react to pre-set trigger conditions in a timely manner. For example, in network management systems, we may want to quickly divert traffic from certain area once a congestion is discovered; in road traffic control, we may want to immediately enact a camera when a car crosses a certain line while the traffic signal is red; and in environmental monitoring, we may want to start taking rain samples soon after certain event happens. In this paper, we tackle this monitoring problem when the data are in the form of streaming time series, and the trigger conditions need to be evaluated continuously.

Consider a stock market monitoring system where each stock  $s^i$  is a streaming time series. A stock broker may register many requests for his/her clients such as "notify Alice whenever stocks  $s^1$  and  $s^2$  are correlated with a correlation value above 0.85 during a one-hour period," and "send information of stock  $s^1$  to Bob whenever  $s^1$  is similar to a stored

pattern segment." The system is to monitor the streaming stock data and trigger the actions in the requests whenever the corresponding conditions are evaluated true.

A perfect system would trigger the corresponding action without any delay whenever a condition becomes true. However, delay is unavoidable in a resource constrained environment. This delay can be alleviated by allowing approximate evaluation and load shedding. In this paper, we define approximate evaluation as to allow some conditions that are actually true (false, resp.) to be reported false (true, resp.), and load shedding as to selectively skip a fraction of conditions from evaluation.

Approximation and load shedding may lead to errors. However, results may still be useful. Approximate methods have appeared in the literature in other contexts as well. For example, a load shedding strategy is deployed in Aurora system [14, 1, 4], and ANN [2] is developed to searching for approximate nearest neighbors.

Approximation can be useful in at least two ways. End users may be satisfied with approximate results, but it is important that users are able to specify their needs on different approximation aspects. Or the system may use approximate results for optimization purposes. For example, a "speculative optimization strategy" can use fast approximate results to prepare (e.g., prefetch) for subsequent complex activities, while precise results can later be used to correct any error made in the aggressive, speculative phase [15]. In this scenario, precision and response time need to be balanced in order to achieve the best overall performance. In this case, a system is the "user", and it is important that the approximate algorithms can satisfy "user" requirements.

Response time, approximation error, and load shedding level are competing factors that need to be balanced by the evaluation system. In this paper, we advocate that it's the users' needs that should ultimately decide how the evaluation system balances them. That is, the evaluation system should satisfy user-specified quality requirements, in terms of these three factors, and a concept of quality of service is needed. The resulting system is called *quality-driven*.

For a quality-driven system, it is important to have the ability to measure the (intermediate) result quality during the evaluation process. However, 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 trigger conditions. So we advocate measuring accuracy in an *a priori* manner, i.e., the false ratios are estimated before the

conditions are evaluated. To do this, we build a prediction model based on historical data analysis. At the evaluation time, we use this prediction model to derive accuracy estimates that are used to guide the evaluation.

Since the evaluation procedure is based on probabilistic prediction models, our system cannot guarantee the satisfaction of accuracy in a strict sense. Instead, similar to the *soft* quality of service (QoS) concept in computer networks [5], 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 users to impose the following quality constraints:

- *Response time constraint*: All evaluation results must be reported within a given time limit.
- *Drop ratio constraint*: The percentage of the conditions that are skipped (thus no evaluation results are reported for them) cannot exceed a given threshold.
- *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).

Ideally, users should be able to impose any combination of the above constraints. But a system may not be able to satisfy all the constraints simultaneously. One way is to ask the users to give up on some constraints (i.e., leave as unconstrained) and let the system to perform its best in terms of these constraints, while satisfying the imposed constraints on the remaining parameters. In our scenario, if one constraint is left unspecified, the system can always satisfy all other constraints. This leads to three possible choices. For each choice, we develop an evaluation algorithm.

This paper makes three contributions. First, we initiate the study of a quality-driven system for evaluating trigger conditions on streaming time series. Second, we show how to use a prediction model to deduce the accuracy estimates. Third, we develop quality-driven evaluation algorithms, and show their effectiveness. Our algorithms also avoid a “starving” phenomenon in which a condition is not being precisely evaluated (instead, it’s either dropped or approximately evaluated) for a long period of time.

## 2. RELATED WORK

The quality-driven aspect of our work is similar to the QoS concept in computer networks [8, 13]. QoS in computer networks allows 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 adopts the QoS concept into trigger condition evaluation on streaming time series and presents a basic design strategy for developing such a quality-driven system.

Aurora [14, 1, 4] 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. The major difference between Aurora and our quality-driven evaluation system is how the quality measures are used to guide the evaluation. In Aurora, the quality metric of an application contributes to an overall QoS function, and the system

tries to maximize the value of this function when it makes scheduling and load-shedding decisions. In our system, we allow multiple QoS parameters, and the system only optimizes an unconstrained quality parameter while satisfying users-imposed constraints on all the remaining parameters.

With limited resources, using approximation techniques in processing continuous queries on data streams has been studied in [7, 9, 6, 12]. However, most approximate evaluation strategies, and even some precise evaluation strategies, only consider one quality aspect and neglect the others. For example, Chain [3] 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 imposed by different users.

## 3. PRELIMINARY

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  $i$ . In this paper, we assume that all streams are synchronized, that is, each stream has a new value available at the same time position.

In general, a trigger condition can be any user-defined predicate on streaming time series, and needs to be evaluated after every data arrival. 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  $C_F = \{c_i | c_i \in C \text{ and } r(c_i, t) = \text{False}\}$  and  $C_D = \{c_i | c_i \in C \text{ and } c_i \text{ is dropped at time position } t\}$ . We call  $C_T$ ,  $C_F$  and  $C_D$  the reported-True set, reported-False set, and dropped set, respectively.

Using the above notation, we define four parameters:

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. *Drop Ratio*,  $DR$ , is the fraction of the conditions (among all the conditions in  $C$ ) that are dropped (not reported).
3. *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.
4. *False Negative Ratio*,  $FNR$ , of a reported-False set  $C_F$  is similarly defined.

Note that response time is defined only on conditions that are reported true.

<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.

## 4. PREDICTION MODEL

While it is easy and straightforward to measure the quality parameters  $RT$  and  $DR$  at any time, it is difficult to measure the other two quality parameters,  $FPR$  and  $FNR$ , without actually evaluating all the conditions. A practical solution 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.

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,

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

In this paper, we make the simplifying assumption that all  $X_i$ 's (for  $1 \leq i \leq n$ ) are mutually independent. Clearly, the mean  $\rho_i$  is also the expected value of  $X_i$ . 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. Specifically,  $\hat{\rho}_i$  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, and  $N$  the sample size.) We may obtain the above by analyzing the historical evaluation results for each condition. We can adopt a data mining approach. Examples can be found in [10, 15]. (3) Too little historical data to estimate  $\rho_i$ . For all the three cases, the estimate of  $\rho_i$  can be viewed as a random variable that follows normal distribution as shown below.

Case	$\mu_i$	$\sigma_i^2$	Note
$c_i$ is precisely evaluated	1(or 0)	0	known True or False
$c_i$ is predicted	$\mu_i$	$\frac{(1-\mu_i)\mu_i}{N}$	from $N$ samples
otherwise	0.5	0.25	unknown

With the prediction model, given a reported-True set of size  $m$ , we have:  $E(FPR) \sim Norm(\mu, \sigma^2)$ , 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 above table accordingly. And we can do the same for  $E(FNR)$

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.

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

In addition to  $FPR$ - and  $FNR$ -constraints, which provide an effective way to guarantee the *overall* evaluation quality, we allow the users to specify two constraints for each individual condition  $c_i$  such that  $c_i$  cannot be reported as true (or false) unless  $\mu_i$  is greater (or less) than a given true (or false) quality threshold. For simplicity, we assume that the users use a global threshold of 0.5 for all individual constraints.

## 5. OPTIMIZATION ALGORITHMS

In this section, we develop optimization algorithms for three optimization problems<sup>†</sup>. Each problem requires a different strategy. For simplicity, in all these algorithms, we assume that the precise evaluation of each condition has the same cost. Thus, the response time can be measured by the number of conditions that have been precisely evaluated (to either True or False) before the last condition in the reported-True set is reported.

As mentioned earlier, the user imposes constraints on some quality parameters and leaves one unconstrained. The evaluation system will optimize for the unconstrained parameter while satisfying the constraints on others. In our scenario there are four quality parameters. For the cases that the unconstrained parameter is either  $FPR$  or  $FNR$ , we just give the algorithm for the  $FNR$  unconstrained case since the other one is symmetric. Thus, we study three optimization problems, namely, 1) minimize response time given accuracy requirements and no drop, 2) minimize drop ratio given response time and accuracy requirements, and 3) minimize false negative ratio given response time and false positive error requirements and no drop. We summarize these optimization problems below.

Algorithm	Constraints	Parameter to be minimized
MinResponse	$DR = 0, \tau_{FPR}, \tau_{FNR}$	$RT$
MinDrop	$RT < \theta_{RT}, \tau_{FPR}, \tau_{FNR}$	$DR$
MinFNR	$DR = 0, RT < \theta_{RT}, \tau_{FPR}$	$FNR$

In each optimization, minimizing a parameter value implies this parameter is unconstrained and the corresponding algorithm will try to minimize its value. Constraints on other parameters (not to be minimized) must be satisfied.

### Algorithm for minimizing response time

The algorithm is shown in Fig. 1. 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 trigger conditions in  $C_T$ . We use a greedy algorithm for this purpose.

In the algorithm, we need to exam if the  $FPR$ -quality of a set  $C_T = \{c_1, \dots, c_m\}$  satisfies a given  $FPR$ -constraint  $\tau_{FPR}$  (using our predicted  $\mu_i$  values). Since we already know that the expected value of  $FPR$  follows normal distribution, we can easily get the confidence  $\beta_T$  by using the standard normal distribution function  $\eta$ <sup>‡</sup>:

$$\beta_T = \eta(z_T), \quad \text{where } z_T = \frac{\sum_{i=1}^m (\theta_E - (1 - \mu_i))}{\sqrt{\sum_{i=1}^m \sigma_i^2}}.$$

And  $z_T$  value can be calculated incrementally.

In the algorithm,  $\mu$ List starts as the list that contains all the conditions in  $C$  arranged in the order, say  $c_1, \dots, c_n$ , such that  $\mu_1 \geq \dots \geq \mu_n$ . The algorithm starts with using **InitExpand** $C_T$  to get initial report-True set  $C_T$ . (This **InitExpand** $C_T$  procedure obtains, without evaluating any conditions, an initial set of conditions that can be reported true without violating the user requirements. See [11] for details.) The trigger conditions in  $C_T$  are reported to be True. Both the  $FPR$ -constraint  $\tau_{FPR}$  and  $\mu > 0.5$  are satisfied by  $C_T$  by the property of **InitExpand** $C_T$ . These are the trigger conditions that can be reported True without doing any precise evaluation. This is Step 1.

<sup>†</sup>In this abstract, we will concentrate on the algorithm for one of the problems. See [11] for the other algorithms.

<sup>‡</sup> $\eta(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$

Consts.:	no drop and satisfy $\tau_{FPR}$ and $\tau_{FNR}$
Goal:	minimize response time ( $RT$ )
Step 1.	Form and report the reported-True set: Initialize $z_T = 0$ and $z_F = 0$ ; $[z_T, i_T] = \text{InitExpand}C_T(z_T)$ ; 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.; - $[z_T, \Delta_T] = \text{Expand}C_T(z_T, i_T + 1)$ - 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: $[z_F, i_F] = \text{InitExpand}C_F(z_F)$
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 $[z_F, \Delta_F] = \text{Expand}C_F(z_F, i_F)$ ; - 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 1: Algorithm MinResponse.

After Step 1, we need to precisely evaluate trigger 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 highest  $\mu$  value. This is the one immediately after the conditions in the initial  $C_T$ . 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 Procedure  $\text{Expand}C_T$ ), 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.

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 trigger 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 trigger 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).

After Step 3, if we still have trigger 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  $C_F$ .

## 6. EXPERIMENTAL RESULTS

In this section, we present our experimental results<sup>§</sup>.

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

A prediction model for each condition is built based on the method in [10] on the data sets generated above. 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.

**Performance parameters:** We use the four quality parameters described in Section 3 (i.e.,  $RT$ ,  $DR$ ,  $FPR$  and  $FNR$ ) to measure the performance of our algorithms. Note that  $DR$ ,  $FPR$  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.

### Results for minimizing response time

This set of experiments is to assess **MinResponse** that minimizes the response time under quality constraints on  $C_T$  and  $C_F$  and no drop allowed.

In this experiment, we set the confidence threshold  $\alpha = 95\%$  for both  $FPR$ - and  $FNR$ -constraints and  $DR = 0$ . 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. 2(a) and (b) show the evaluation quality achieved in terms of actual  $FPR$  and  $FNR$ . The two plots of Fig. 2(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. 2(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 that the average is either below or very close to the corresponding required expected-mean threshold  $\theta_E$  for all the runs.

Fig. 2(c) shows the performance of **MinResponse** in terms of response time. For comparison, a naive algorithm is im-

<sup>§</sup>Again, we only report results on **MinResponse** in this abstract, and the reader is referred to [11] for other results.

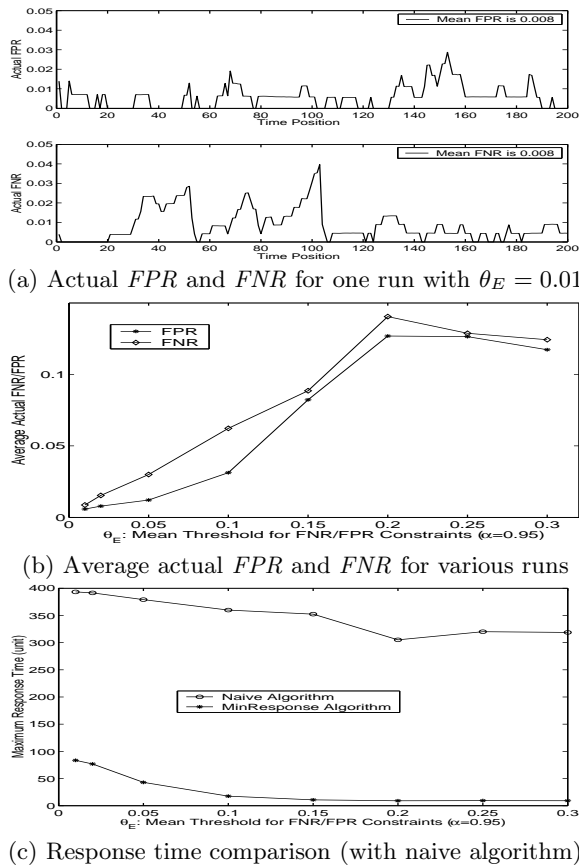


Figure 2: Quality (and performance) of MinResponse.

plemented: 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 MinResponse (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 MinResponse with this naive algorithm for different runs with  $\theta_E$  values in  $[0.01, 0.3]$ . We can see that MinResponse consistently outperforms the naive algorithm. Note that the response time of MinResponse 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 MinResponse is significant. For example, given  $\theta_E = 0.01$ , MinResponse 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.

## 7. CONCLUSION

In this paper, we studied a trigger condition evaluation system that considers user-specified quality requirements. 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 approximate answers. Based on this prediction method, we designed algorithms for three different optimization problems. Our experiments showed that the algorithms are effective in reaching corresponding optimization goals.

Note in our system, when a condition has not been pre-

cisely evaluated for a few steps (either due to load shedding or approximation), its expected result will become more uncertain ( $\mu$  value goes closer to 0.5) because of the nature of our prediction model. Thus, eventually, this condition will have to be precisely evaluated. This automatic adjustment can avoid each condition from starving of precise evaluation.

It will be interesting to see how our quality-driven strategy works on other optimization problems. For example, we may optimize in a more global manner. The situation to consider may be to combine parameters into a global function for optimization, or to consider trigger conditions from a few consecutive basic windows and derive better quality in a more global sense.

## 8. REFERENCES

- [1] D. J. Abadi et. al. Aurora: A data stream management system. In *SIGMOD*, 2003.
- [2] 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.
- [3] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator scheduling for memory minimization in data stream systems. *SIGMOD*, pp. 253–264, 2003.
- [4] D. Carney et. al. Monitoring streams - a new class of data management applications. In *VLDB*, 2002.
- [5] 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.
- [6] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, pages 40–51, 2003.
- [7] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, pages 61–72, 2002.
- [8] P. Ferguson and G. Huston. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley & Sons, 1998.
- [9] S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *SIGMOD*, pages 265–276, 2003.
- [10] 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.
- [11] L. Gao, M. Wang, and X. S. Wang. Quality-driven evaluation of trigger conditions on streaming time series. CS Technical Report, University of Vermont, 2004.
- [12] 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.
- [13] D. McDysan. *QoS and Traffic Management in IP and ATM Networks*. McGraw-Hill Osborne Media, 1999.
- [14] 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.
- [15] X. Sean Wang, L. Gao, and M. Wang. Condition Evaluation for Speculative Systems: a Streaming Time Series Case. *STDBM 2004* pp. 65-72.