A Learning-Based Approach to Estimate Statistics of Operators in Continuous Queries: a Case Study

Like Gao* Min Wang† X. Sean Wang* Sriram Padmanabhan†
*Dept. of Information and Software Engineering, George Mason University, VA
{lgao, xywang}@gmu.edu
†IBM T.J. Watson Research Center, NY, {min, srp}@us.ibm.com

ABSTRACT

Statistic estimation such as output size estimation of operators is a well-studied subject in the database research community, mainly for the purpose of query optimization. The assumption, however, is that queries are ad-hoc and therefore the emphasis has been on capturing the data distribution. When long standing continuous queries on a changing database are concerned, a more direct approach, namely building an estimation model for each operator, is possible. In this paper, we propose a novel learning-based method. Our method consists of two steps. The first step is to design a dedicated feature extraction algorithm that can be used incrementally to obtain feature values from the underlying data. The second step is to use a data mining algorithm to generate an estimation model based on the feature values extracted from the historical data. To illustrate the approach, this paper studies the case of similarity-based searches over streaming time series. Experimental results show this approach provides accurate statistic estimates with a low overhead.

1. INTRODUCTION

Long standing queries, also called continuous queries [28], are those that are typically issued once and then evaluated continuously, every time when the database is updated or some pre-defined events occur. These queries appear in a wide range of applications, especially in applications that need to monitor certain events from data sources. The database research community has shown great interest in these queries over the last decade [6; 18]. Recently, this interest has increased sharply due to the emerging needs of data stream management [3; 9; 10; 11; 19].

As in traditional database systems, it is important to estimate related statistics of the operators used in a continuous query. For example, in cost-based query optimization, an optimizer needs to estimate the cost and output size of operators to determine the most efficient execution plan. In continuous queries, these statistic estimates are more crucial since the database is usually updated frequently and fast query response is required.

A variety of techniques exist to estimate statistics of operators. Among all the statistics, output size estimation has attracted the most attention. Methods in the literature include non-parameric methods (including histograms [15; 21]), parametric methods [25], curve fitting [26], sampling [16], and methods based on query feedback [5]. Among them, histogram method is most commonly used in database systems due to its computational efficiency and independence of data distribution. However, despite technical differences, a common feature of these techniques is that they all aim at capturing the underlying data distribution as precisely as possible under certain storage constraint. Such captured data distributions are then used to estimate the output sizes (selectivities) of operators.

When dealing with continuous queries, we may use a different approach due to the difference between a traditional query and a continuous query. In a traditional query, the database is assumed to be static and the queries are ad-hoc, i.e., the system needs to handle any possible query. This is why most existing techniques focus on capturing the entire underlying data distribution. In a continuous query, however, the query is long standing and the database changes each time the query is evaluated. To estimate the statistics of a given operator in a continuous query, we only need to know the part of the data that is relevant to the specific operator (instead of the entire underlying data distribution) and capture the evolution of it when the database changes. This leads to a different estimation method that does not require the entire data distribution.

In this paper, we reconsider the estimation problem in the context of continuous queries. Also, we do not limit this study to standard operators (e.g., selection and join) and the standard statistics (e.g., output size). Consider the problem of estimating statistics of an operator O in a continuous query, where the query is fixed and the database is changing. The fixed query assumption implies that the output statistic, say $stat_o$, can solely be determined by the input data D, say $stat_o = f_o(D)$, where f_o is a fixed estimation function for operator O. In other words, we only need to know the input data to obtain the statistic directly. In contrast, most existing estimation methods capture the data distribution in advance and determine the statistic of a specific operator at the query evaluation time using the data distribution.

The advantage of this direct method is its potential to get more accurate estimates. However, the continuous query may involve a large volume of input data and data types can be complex. It is not clear how to establish the estimation function f_o directly. In this paper, we decompose the estimation function f_o into two components, namely *fea-*

ture extractor and *statistic estimator*. More specifically, let $stat_o = s_o(e_o(D))$, where e_o is the feature extractor that obtains feature values from the underlying data, and s_o is a statistic estimator that uses the obtained feature values as input. The feature extractor not only reduces the data volume, but also extracts the relevant part of the underlying data. In addition, it converts the underlying data into a data type that will be used for building the statistic estimator. In some cases, the extractor can work incrementally when database is updated to increase the efficiency. The statistic estimator can be built through the use of traditional methods, such as decision tree algorithms, polynomial functions and even histograms.

The feature extractor is usually determined manually and in many cases, a dedicated incremental procedure is developed to obtain feature values in order to reduce the overhead. Strategies to establish the statistic estimator can be classified into two categories: analytic approaches and empirical approaches. In an analytic approach, by analyzing the underlying evaluation procedure of an operator, we may be able to derive the statistic estimator which takes the features as input. However, when the operator is complicated or it involves multiple resources, an empirical approach may be used to establish the statistic estimator, if enough experimental data, i.e., a collection of feature data and corresponding actual statistics, are available. For complex situation, the empirical approach is more suitable because we can directly use historical evaluation data of the continuous query as experimental data, and apply an available data mining algorithm to analyze these data to build the statistic estimator. This is why we call our proposed approach *a learning-based* one and the statistic estimator is also called *estimation model*.

To validate our approach, we study a special kind of continuous query, namely similarity-based search over streaming time series. We design a feature extractor by using data approximations via DFT (Discrete Fourier Transform), and use a decision tree algorithm to establish the estimation model. We then use experiments to assess the performance of our proposed approach. The experimental results show that this approach provides accurate estimates with a low overhead.

The rest of this paper is organized as follows. We introduce the similarity-based search over streaming time series in Section 2 and describe two important statistics in Section 3. We present the detailed learning-based approach in Section 4. In Section 5, we report our experimental results and, in Section 6, give the related work. We conclude this paper in Section 7.

2. SIMILARITY-BASED SEARCH OVER STREAMING TIME SERIES

In this section, we define the notion of a similarity-based search over streaming time series.

There are two types of data involved in such a search, namely *time series* and *streaming time series*. A *time series*, denoted as $\mathsf{P} = \langle p_1, p_2, \ldots, p_n \rangle$, is a finite sequence of n real numbers, with its length n denoted $len(\mathsf{P})$. A *streaming time series*, $\mathsf{S} = \langle s_1, \ldots, s_{t-1}, s_t, \ldots \rangle$, is an infinite sequence of real numbers with the assumption that its values arrive at the database system sequentially. The subscript t indicates the data arrival time of s_t. Given a streaming time series S, the *streaming time series* S *up to time* t denotes the subseries of S before time t, inclusively, i.e., $\langle s_1, \ldots, s_{t-1}, s_t \rangle$.

Given two time series, there are different ways to measure the similarity between them [12]. In this paper, the weighted Euclidean distance,

$$sim(\mathsf{Q}, \mathsf{P}) = \sqrt{\sum_{i=0}^{N-1} (q_{len(\mathsf{Q})-i} - p_{len(\mathsf{P})-i})^2 / N},$$

where $N = min(len(\mathsf{Q}), len(\mathsf{P}))$, is used to define the similarity between time series Q and P. More specifically, this measure is defined on the common N-suffix of two time series, where N is the length of the shorter time series. Under this definition, the more similar the two time series, the smaller the distance between them. This definition can be applied directly to a time series and a streaming time series up to current time without any modification. [1]

Assume Q is a given query time series, $\mathcal{P}$ is a set of time series $\{\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_m\}$ where each P_i ($1 \leq i \leq m$) in $\mathcal{P}$ is called a pattern, an integer $k > 0$ is called *similarity rank* and a real number $\alpha \geq 0$ is called *similarity threshold*. A pattern P_i in $\mathcal{P}$ is a *k-nearest & α-near neighbor* of Q if there exist at most $k-1$ patterns $\mathsf{P}_j \in \mathcal{P}$ such that $sim(\mathsf{Q}, \mathsf{P}_i) > sim(\mathsf{Q}, \mathsf{P}_j)$, and $sim(\mathsf{Q}, \mathsf{P}_i) \leq \alpha$. A k-nearest & α-near neighbor is also called a k-α-near neighbor for short. When α or k is set to some special value, the k-α-near neighbor is exactly the *k-nearest neighbor* (when $\alpha = \infty$) or the *α-near neighbor* (when $k = \infty$).

Definition Given a streaming time series S, a set of pattern time series $\mathcal{P}$, a similarity rank k and a threshold α, the *similarity-based search over stream* S is to find, at each data arrival time t, all k-α-near neighbors of S up to time t from set $\mathcal{P}$.

A continuous query may contain one or more similarity-based searches as its operators. The query optimizer needs to find the most efficient query execution plan so that the system can finish the query evaluation as soon as possible. It is important to provide accurate statistic estimates of each involved similarity-based search to the optimizer.

3. TWO IMPORTANT STATISTICS FOR OPTIMIZING CONTINUOUS QUERIES

In the following, we describe two important statistics that are useful in optimizing a continuous query, namely *cost* and *emptiness* of a similarity-based search. Cost is the execution time of the search, and emptiness is a boolean value to state the search result size being zero or not.

Consider a continuous query that contains the conjunction of two predicates, each involving a similarity-based search. For example, predicate p_1 is to test if a streaming time series

[1]Note our definition of *weighted Euclidean distance* is different from that in [14], where the two time series involved in the distance definition have finite length.

S_1 has a nearest neighbor that is within a given threshold in pattern set $\mathcal{P}_1$ at the current time position.

It is easy to see that the optimal evaluation order of these two predicates can be determined as follows: 1) If the two similarity-based searches have similar evaluation cost and one of them is more likely to result in an empty set (thus the corresponding predicate is false), this search should be evaluated first; 2) If the two similarity-based searches are equally likely to generate an empty set, the one that is less costly should be evaluate first.

From this simple example, we can see that the optimal plan is determined by the cost and emptiness of the two searches. We thus choose cost and emptiness of a similarity-based search as the statistics to be studied and see how to estimate their values.

We do not study output size of a similarity-based search in this paper, although it is one of the most important statistics in query optimization with relational operators. This follows from two observations in a similarity-based search. Firstly, a small similarity rank k may be given, e.g., $k = 1$. This implies that the output size is usually not greater than this value k. Secondly, an optimizer may only need to know the emptiness of a search, and does not care about the actual result size. This happens in many applications with continuous queries, such as monitoring streams for abnormal event detection. Therefore, we decide to study emptiness instead of output size in this paper.

Although we do not study how to estimate the output size of a similarity-based search, we can still apply the strategies discussed hereafter for output size estimation.

4. STATISTIC ESTIMATION

As mentioned in the introduction, we need to consider the execution procedure of the operator to determine the feature extractor. In this paper, we assume a specific procedure is used to evaluate the similarity-based searches. By considering this procedure, we present a strategy below to design the feature extractors. We then describe how to use decision tree methods to establish the estimation models, and discuss how to use these models to obtain the statistic estimates.

The specific procedure for similarity-based search is a direct computation method to find k-α-near neighbors. More specifically, in this method, we first determine a number of pattern time series where the k-α-near neighbors must be within them. We then calculate the distance from each of these patterns to the streaming time series by directly applying the similarity formula. Having all these distances, we can quickly determine which patterns are the actual k-α-near neighbors.

4.1 Feature Extractors

Feature extractor obtains feature values from the underlying data with the goal of minimizing estimation overhead. In order to reduce the overhead, we develop an incremental approach to design the feature extractor by using data approximations of time series and streaming time series.

We first describe how to approximate time series and streaming time series. The approximation method that we use in this paper is similar to that in [8], where a small number of significant DFT coefficients are used to represent a time series or a subseries of a streaming time series. We call these coefficients as the *approximations via DFT*.

For the static pattern time series, i.e., all patterns are stored in a database and do not change during the continuous query evaluation, we can pre-process them to find their approximations off-line. Since patterns in the database may have different lengths, we need to use DFT with the corresponding length to perform the approximation for each pattern.

For a streaming time series, we can only obtain its approximation in an on-line fashion when each new data value arrives. To reduce the overhead of computing DFT at each time position, we use an incremental approach to get the approximation of the N-suffix of the streaming time series, where N is the length of a pattern time series. Specifically, we use Sliding DFT [27], given as the following proposition, to quickly calculate the DFT coefficients. We can see both the space requirement and computation cost of this incremental DFT calculation are very small.

Proposition Given a DFT length N and a streaming time series $\mathsf{S} = \langle s_1, \ldots \rangle$, let $Y = \langle y_0, y_1, \ldots, y_{N-1} \rangle$ be the DFT coefficients of the N-suffix of S at time $t-1$, and $Y' = \langle y'_0, y'_1, \ldots, y'_{N-1} \rangle$ be those coefficients of S at time t, then Y' can be found by $y'_n = e^{j2\pi n/N}[y_n + (s_t - s_{t-N})/N]$ for $n = 0 \ldots N-1$.

Two facts should be noticed about the approximation. The first is that corresponding to each distinct pattern length, there is an approximation of the streaming time series. This means that a streaming time series may have multiple versions of approximation at any given time position. The second is that we can calculate a distance lower bound for two time series using their approximations via DFT, as given in the following proposition.

Proposition For two time series Q and P where $N = min(len(\mathsf{Q}), len(\mathsf{P}))$, if let $\hat{\mathsf{Q}}$ and $\hat{\mathsf{P}}$ denote the approximations of the N-suffix of Q and P, respectively, and let n denote the dimensionality of the approximations, then

$$\sqrt{\frac{n}{N}} sim(\hat{\mathsf{Q}}, \hat{\mathsf{P}}) \leq sim(\mathsf{Q}, \mathsf{P}).$$

Proof: Let the N-suffix of Q and P be $\langle x_1, x_2, \ldots, x_N \rangle$ and $\langle y_1, y_2, \ldots, y_N \rangle$, respectively. Let their corresponding DFT coefficients Q_N and P_N be $\langle X_1, X_2, \ldots, X_N \rangle$ and $\langle Y_1, Y_2, \ldots, Y_N \rangle$, respectively. Using Parseval's Theorem [22], we have:

$$sim(\mathsf{Q}, \mathsf{P}) = \sqrt{\frac{\sum_{i=1}^{N} (x_i - y_i)^2}{N}} = \sqrt{\frac{\sum_{i=1}^{N} |X_i - Y_i|^2}{N}}.$$

For the approximations $\hat{\mathsf{Q}}$ and $\hat{\mathsf{P}}$ with dimensionality of n, $n \leq N$, we have

$$sim(\hat{\mathsf{Q}}, \hat{\mathsf{P}}) = \sqrt{\frac{\sum_{i=1}^{n} |X_{mi} - Y_{mi}|^2}{n}} \leq \sqrt{\frac{\sum_{i=1}^{N} |X_i - Y_i|^2}{n}}.$$

The proposition follows. $\square$

In this paper, to simplify the illustration, we only use the first DFT coefficient to approximate the pattern time series and streaming time series, i.e., $n = 1$ in the above inequation.

We now consider how to extract feature values for cost estimation by using these approximations. First, we pre-process the pattern time series by sorting all of them by their approximations (the first DFT coefficients) in increasing order, and we re-assign the corresponding ranks as their new indices. We perform this task off-line since no streaming time series is involved. We then determine as features two pattern indices, namely $LowerIndex$ and $UpperIndex$, whenever a new value of the streaming time series arrives. These two indices determine a range of patterns. The reason that we use these two indices as features is because the number of patterns in the range is closely related to cost of the similarity-based search.

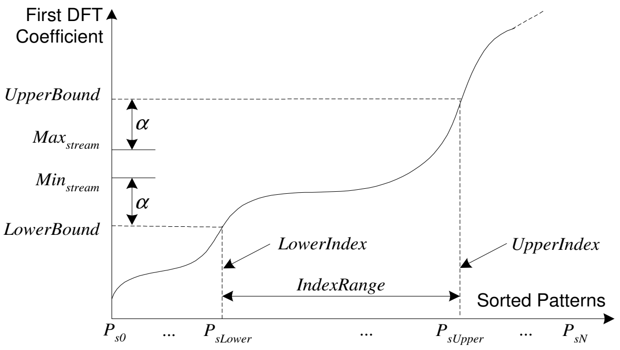

Figure 1: Cost feature extractor.

In Figure 1, we illustrate the detailed steps of obtaining these two feature values in an on-line fashion. In this figure, the x-axis represents the indices (ranks) of the pattern time series, the y-axis represents the approximation values and the monotonic increasing curve represents the first DFT coefficients of all patterns in the pattern set after we perform the sorting. We carry out the feature extraction as follows. First, we use the incremental DFT method to get multiple approximations of the streaming time series up to current time. (Each approximation corresponds to a distinct length for the pattern in the pattern set.) Assume these approximations have a minimum value of Min_{stream} and a maximum value of Max_{stream}. Now we can determine a distance range by computing $LowerBound = Min_{stream} - \alpha$ and $UpperBound = Max_{stream} + \alpha$. Assume p is a pattern in the pattern set and p is an α-near neighbor of the streaming time series at current time position. Based on the proposition on distance lower bound, we can see that the first DFT coefficient of p must be in the range of $[LowerBound, UpperBound]$. Finally, we use binary search to quickly find the corresponding index range $[LowerIndex, UpperIndex]$.

We use the similar procedure to extract features for the emptiness estimation. We also choose these two pattern indices as features and also use the process in Figure 1 to get their values, since the emptiness of the search should be closely related to the index range. In addition, we consider a unique property, namely continuity [10] of a similarity-based search over streaming time series. This property says that successive evaluations of a search are likely to result in the same output. This implies that if one evaluation of the search is empty (non-empty), then the following one is also likely to be empty (non-empty). Hence, we can use previous evaluation emptiness to estimate for current one. So besides the above two pattern indices, we also use a number of most recent emptiness as features.

In summary, we use as features two pattern indices and the emptiness of a number of previous evaluations. In the implementation, we use an additional feature, $IndexRange = UpperIndex - LowerIndex$. Although this feature seems to be redundant, it does make the estimation models more concise and more accurate. We tabulate all the features for each extractor in Figure 2.

Extractor	Features
Cost	$LowerIndex$, $UpperIndex$ and $IndexRange$
Emptiness	$LowerIndex$, $UpperIndex$, $IndexRange$ and a number of previous $Emptiness$

Figure 2: Features used.

4.2 Estimation Models

Building and using estimation models are straightforward. It is a typical data mining application.

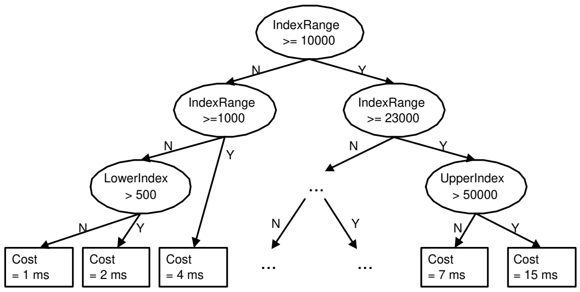

Figure 3: A sample cost estimation model

We first look at how to establish the estimation models. We first run the similarity-based search for a sufficient number of times as the streaming time series come in. During each evaluation, we use feature extractors to get the feature values and collect the actual evaluation cost and emptiness. Having these collected data as training data set, we then apply data mining algorithms such as C4.5 decision tree [23] to establish the estimation models. We give a sample decision tree for the cost estimation in Figure 3.

Using the estimation models to estimate the statistics is also straightforward. When new values of the streaming time series arrive, we use the feature extractors incrementally to obtain the feature values, and then feed them into corresponding estimation models to get the estimates.

5. EXPERIMENTAL RESULTS

In this section, we report our experimental results that demonstrate the effectiveness of the proposed learning-based approach. The experiment is performed on a desktop box with Pentium III 1.2GHz CPU and 512M memory, running Windows XP Professional, and the code is written in programming language C++. The data mining algorithm used to build estimation models is C4.5 decision tree algorithm [23]. All datasets are loaded into memory in advance and hence the cost measured only refers to the computational cost.

Datasets
Both pattern time series and the streaming time series are synthetic data. The 10^5 pattern time series are generated independently via a random-walk function. Each pattern has a length between 50 and 300. The streaming time series is constructed by randomly picking up some pattern time series from the above and concatenating them into one long sequence. In order to fill the gap between two successive patterns, some values are interpolated in between via PCHIP (Piecewise Cubic Hermite Interpolating Polynomial) functions. In addition, some patterns appear more often than others in order to simulate periodic phenomena in real world streams. Finally, white noise is added to the streaming time series.

Pattern time series are pre-processed to obtain their approximations via DFT and they are sorted by their first DFT coefficients. The approximations of streaming time series are calculated on fly with the incremental DFT approach given in Section 4.1. We will only use the first DFT coefficients of streaming time series and pattern time series to derive the features.

Performance measures
The performance (accuracy) of cost and emptiness estimation models is given as follows:

- The accuracy of the cost estimation model: the ratio that the estimated cost is within a given tolerance of the actual cost [2].
- The accuracy of the emptiness estimation model: the ratio that the estimation model correctly predicts the emptiness.

Continuous queries
We use similarity-based searches that ask for 1-α-near neighbors of the streaming time series as the continuous query, where α varies from 0.05 to 0.1, step by step. Here we fix the similarity rank $k = 1$ for two reasons. The first is that a small change of k does not make much difference to the evaluation cost. This can be seen from our feature extraction procedure, where only α is used. The second is that emptiness estimation of a search is independent of the rank value k. Indeed, if and only if a 1-α-near neighbor search results in an empty (non-empty) set, any k-α-near neighbor search will also result in an empty (non-empty) set.

Experimental results
In the experiments, we evaluate the above similarity-based search for 2,000 time positions, which we call a *run*. For each run, we choose the similarity threshold α between 0.05 and 0.1. In each run, we collect feature values, actual cost and emptiness of the search at each time position. We then apply the C4.5 algorithm on 70% of the collected data to find out the two estimation models. We use the remaining 30% data to test the accuracy of the estimation models. The results are given in Figure 4.

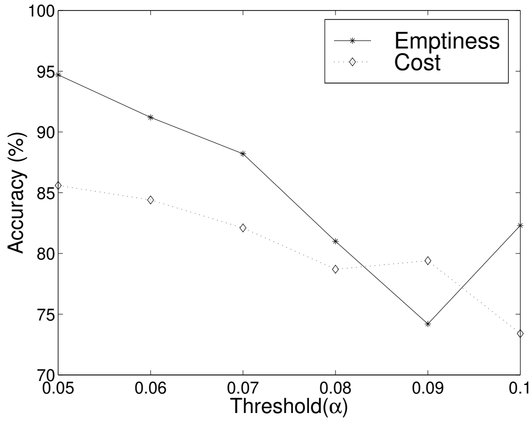

Figure 4: Performance of the models.

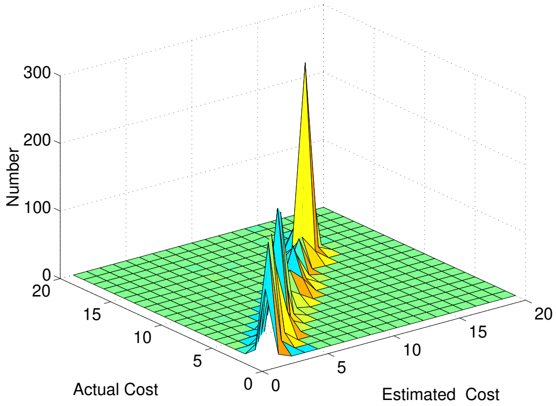

Figure 5: Distribution of estimation (cost).

From Figure 4, we can see that the emptiness estimation model has a relatively high accuracy. An interesting trend is that as the threshold increases, the accuracy goes down first and then goes up. This follows the fact that when threshold is very small (very big), the search is very likely to result in an empty (non-empty) output. In these cases, the estimation model can accurately estimate the emptiness. When the threshold takes values in between, the emptiness is more sensitive to the feature values. In this case, the estimation model has a relatively low accuracy.

The major reason for performance drop is that we only use one DFT coefficient to derive features. This introduces errors to the feature values. These errors have different impact on estimate accuracy for different similarity threshold values. When the impact becomes large, the accuracy of the estimation drops and so does the performance.

The accuracy of the cost estimation model is around 70% to 80%, as shown in Figure 4. Note that a mis-classified cost

[2] The tolerance is 1ms in the experiments reported, which is less 10% of the average cost.

does not necessarily mean a bad estimate, since it may be still very close to the actual cost. To assess this, we run the similarity-based search with threshold $\alpha = 0.1$ for another 2,000 time positions, and study the cost estimation distribution. The result is shown in Figure 5. In this figure, each pair of actual cost and estimated cost corresponds to one evaluation. The vertical value indicates the total number of times that a pair of estimated cost and actual cost appears. Since all non-zero height values are clustered to the diagonal in this figure, this means that in each evaluation, the estimated cost is very close to the actual cost.

For the search with $\alpha = 1$, the averaged time to find the two statistic estimates is 0.05ms. Compared to the overall search time of 15ms on average, this is a very small overhead. Experiments with other thresholds also confirm that our proposed approach has very low overhead.

6. RELATED WORK

Continuous queries have long been attracting the attention of database researchers. Terry et al. [28] perhaps were the first one to introduce the notion of "continuous queries" for a class of queries that are issued once and then run "continuously" over databases. Later Liu et al. [17; 18] also presented a similar concept, termed "continual query". Recently, due to the emerging needs of data stream management systems, the interest in continuous queries, especially in continuous queries over streams, has increased sharply. One direction in this field is to design stream data process systems that support continuous queries, i.e., Chen et al. presented the design of the NiagraCQ system [6; 7]; Babu and Widom [4], Madden and Franklin [19] also reported system architecture and related issues for dealing with continuous queries. Another direction is query plan optimization for continuous queries [20; 29]. Some more related details of continuous queries and streams can be found in the tutorial [3; 11].

In this paper, we studied the estimation problem for similarity-based searches over streaming time series. Most previous work [1; 2; 24] on similarity-based searches was concentrated on ad-hoc queries over stationary data sources, and in most cases, the time series stored in databases are supposed to have the same length. Although [9; 10] also study similarity-based searches over streaming time series, their main contribution is on how to efficiently evaluate similarity-based search on a single streaming time series. In contrast, this paper considers the scenario that a continuous query involves multiple streaming time series and multiple similarity-based searches, and focuses on estimating statistics that are necessary for optimizing such a query.

This paper is also closely related to the literature on statistic estimation, which has been addressed in the introduction section. Besides, another interesting work [13] used machine learning algorithms on the training queries to "re-vitalize" the existing output size estimation methods whose performance previously deteriorated due to high update loads. However, all these work is based on modeling the data distribution. In contrast, in this paper, our learning-based approach is a direct estimation method without the step of modeling data distribution.

7. CONCLUSIONS

In this paper, we introduced a learning-based framework to estimate statistics of an operator in a continuous queries. The framework consists of extracting features and building learning model using a training workload. We studied the case of similarity-based query over streaming time series to validate this approach. Our experiments demonstrated the effectiveness of our proposed approach.

It should be noted that there may be different ways to design a feature extractor. It is important for a feature extractor to be well constructed so that the chosen method can lead to accurate estimation models. It will be interesting to study the strategy on how to find an "optimal" pair of feature extractor and estimation model, which may yield the best estimation accuracy under given resource constraint. Another interesting research direction is to apply this learning-based approach to other types of continuous queries.

8. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Procs. of FODO*, pages 69–84, 1993.

[2] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *The VLDB Journal*, pages 490–501, 1995.

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Procs. of PODS*, pages 1–16, 2002.

[4] S. Babu and J. Widom. Continuous queries over data streams. In *SIGMOD Record, Sept. 2001*, 2001.

[5] Chungmin Melvin Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In *Procs. of ACM-SIGMOD*, pages 161–172, 1994.

[6] J. Chen, D. J. DeWitt, and J. F. Naughton. Design and evaluation of alternative selection placement strategies in optimizing continuous queries. In *ICDE Conference*, 2002.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Procs. of ACM-SIGMOD*, pages 379–390, 2000.

[8] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Procs. of ACM-SIGMOD*, pages 419–429, 1994.

[9] L. Gao and X. S. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *Procs. of ACM-SIGMOD*, 2002.

[10] L. Gao, X. S. Wang, and Z. Yao. Evaluating continuous nearest neighbor queries for streaming time series via pre-fetching. In *CIKM*, pages 485–492, 2002.

[11] Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Querying and mining data streams: You only get one look. In *Procs. of ACM-SIGMOD*, 2002.

[12] D. Gunopulos and G. Das. Time series similarity measures. In *Procs. of KDD*, pages 243–307, 2000.

[13] Banchong Harangsri, John Shepherd, and Anne H. H. Ngu. Query size estimation using machine learning. In *Database Systems for Advanced Applications*, pages 97–106, 1997.

[14] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2000.

[15] Robert Kooi. *The Optimization of Queries in Relational Databases*. PhD thesis, Case Western Reserve University, 1980.

[16] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In *Procs. of ACM-SIGMOD*, pages 1–11, 1990.

[17] L. Liu, C. Pu, R. S. Barga, and T. Zhou. Differential evaluation of continual queries. In *International Conference on Distributed Computing Systems*, pages 458–465, 1996.

[18] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE TKDE*, 11(4):610–628, 1999.

[19] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE Conference*, 2002.

[20] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Procs. of ACM-SIGMOD*, pages 49–60, 2002.

[21] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-based histograms for selectivity estimation. In *Procs. of ACM-SIGMOD*, pages 448–459, 1998.

[22] A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing*. Prentice-Hall, Inc, 1975.

[23] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgann Kaufmann, 1993.

[24] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *Procs. of ACM-SIGMOD*, pages 13–25, 1997.

[25] P.G. Selinger, M.M. Astrahan, D.D. Chainberlin, R.A. Lorie, and T.G. Price. Access path selection in a relational database management system. In *Procs. of ACM-SIGMOD*, pages 23–34, 1979.

[26] W. Sun, Y. Ling, N. Rishe, and Y. Deng. An instant and accurate size estimation method for joins and selection in a retrieval-intensive environment. In *Procs. of ACM-SIGMOD*, pages 79–88, 1993.

[27] Jeffrey D. Taft. The Sliding DFT Page. On-line. http://www.nauticom.net/www/jdtaft/DFT_increm.htm.

[28] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *Procs. of ACM-SIGMOD*, pages 321–330, 1992.

[29] S. D. Viglas and Jeffrey F. Naughton. Rate-based query optimization for streaming information sources. In *Procs. of ACM-SIGMOD*, pages 37–48, 2002.