Real-Time Indexing of Complex Data Streams

Petr Chmelar, Michal Drozd, Michal Sebek and Jaroslav Zendulka
IT4Innovations Centre of Excellence, Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
{chmelarp, idrozd, isebek, zendulka}@fit.vutbr.cz

Abstract — **The paper deals with indexing of a complex type data stream stored in a database. We present a novel indexing schema and framework referred to as ReTIn (Real-Time Indexing), the objective of which is to allow indexing of complex data arriving as a stream to a database with respect to soft real-time constraints met with some level of confidence for the maximum duration of insert and select operations. The idea of ReTIn is a combination of a sequential access to the most recent data and an index-based access to less recent data stored in the database. The collection of statistics makes balancing of indexed and unindexed parts of the database efficient. We have implemented ReTIn using PostgreSQL DBMS and its GIN index. Experimental results presented in the paper demonstrate some properties and advantages of our approach.**

I. INTRODUCTION

During last three decades data stream sources and data-intensive applications has appeared [1]. In contrast to traditional databases where data is stored in finite data sets, a data stream is a continuous, possibly infinite stream of changing and often high dimensional data that must be processed under some real-time constrains usually. Examples of such applications include network monitoring, financial and security, surveillance, sensor networks and other applications processing temporal data. Although research in real-time database systems received a lot of attention in last two decades, the primary objective of the real-time support in these databases was different compared to data streams [2]. Several basic data stream specific techniques have been developed for continuous querying [3], sliding window query processing [4], approximate query processing [5], sampling, sketching and synopsis construction [6]. Most of these techniques rely on data stream processing in the main memory, however this might be unsatisfactory for applications mentioned.

The goal is to store the data in a database. An index is a data structure designed to increase the data access speed at the expense of decrease the data modification speed. B-tree, hashed or bitmap are not appropriate for high-dimensional data. In addition, costs of updates limit their effective use. The problem of index maintenance is more critical when employed advanced indexing techniques for high-dimensional data as KD-tree, R-tree or inverted index. In such a case, it may be necessary to rebuild the index completely after some time. As a result, some data stream specific indexing methods have been developed. Multi-granularity aggregation indexing [7] is an integrated structure managing summarized information of snapshots. Po-tree [8] is an indexing structure for spatio-temporal databases with soft real time constraints which combines two different structures for spatial and temporal dimensions. However, we haven't found a general approach that satisfies our needs.

Our research in real-time and data stream indexing was motivated by two areas. First is the need to index metadata of moving objects produced by computer vision modules of our experimental surveillance network system SUNAR [9]. The main operation of SUNAR is the persistent tracking of objects moving in a space watched by multiple surveillance cameras. The tracking is based on similarity of values of moving objects characteristics, both spatio-temporal and visual. The intelligent cameras produce a data stream of this kind, which is necessary to index to make the similarity search possible in near real-time. The second domain that has motivated our research is computer security, namely a data stream processed by an intelligent intrusion detection system (IDPS) that monitors and analyzes the computer network traffic in real time. It is based on extended data flow protocol, which includes source and destination IP addresses, timing, packet sizes and signatures of both packets and attacks or another data that must be logged, analyzed and reported as soon as possible.

There are many other application domains that deal with streams of spatio-temporal data representing moving objects. For example, an air-traffic control to support decisions about flight paths and the landing order based on data such as position, altitude, speed or fuel left. There is rarely enough time to re-index the database under special circumstances – as in a geographic information system presented in a case study [8] that stores and evaluates data issued from an array of spatially referenced sensors, used to a natural disaster prevention.

In this paper we present an indexing schema and framework referred to as ReTIn (Real-Time INdexing), the objective of which is to allow indexing of complex data arriving as a stream to a database with respect to soft real-time constraints. A concept of a soft real-time constraint is similar to one known from real-time databases [10]. It is not a hard constraint that has to be always met but the number of its violations must be minimized. The idea of our approach is similar to a real-time index/cache consistency maintenance technique Codir for text retrieval systems presented in [11]. It builds a transient index for new document updates and queries are processed using both permanent and transient index. To minimize performance overhead associated with document database updates, Codir integrates transient index with permanent index lazily using piggybacking [12] for statistics.

Our indexing schema consists of two main parts where data is stored. Values arriving in a data stream are inserted into the unindexed part that contains the most recent data. Less recent data is stored in the other part which is indexed on backgroud.

Queries are processed accessing data in both parts. The schema maintenance, which includes moving data from unindexed part to indexed one, is controlled by two soft real-time constraints for query and insert processing and it uses piggybacking to collect and update statistics. The schema maintenance runs as a background process for all database operations.

The content of the paper is organized as follows. The next section contains problem formulation and describes the structure of the proposed indexing schema and operations on it. Section 4 presents experimental results and Section 5 concludes the work.

II. ReTIn indexing schema concepts

The ReTIn indexing schema supports the most important real-time data stream operations on a single table in the database, in which a portion of the stream is stored – insert and select (executes a query). There are three parameters that control the behavior of the indexing schema: a maximum time of insert operation $T_{I\,MAX}$, a maximum time of select operation $T_{S\,MAX}$ and a confidence factor R. Then the schema meets the constraints $T_{I\,MAX}$ and $T_{S\,MAX}$ as soft constraints with confidence R*σ where σ is a standard deviation of execution times distribution and R a selected confidence factor as described further.

A. Problem formulation

Let *ds* be a data stream of data elements *e* of a type *dt,* which is complex in general – composite and/or multiple-valued. The data stream is processed using a sliding window. Assume that the window is larger than it fits in the main memory. The content of the window is stored in a database table *D*, not necessarily normalized. The size of the window is not specified in advance. Instead, real-time constraints $T_{I\,MAX}$ and $T_{S\,MAX}$ are (user) specified durations of insert and select operations on the table *D*. Thus, the size of the window is dependent on the duration of these operations. It is required to minimize the number of violations of the timing constraints. The softness of the constraints is dependent on the probability of their violation. We can introduce estimates for maximum processing times of insert and select operations (estimated maximum) on *D*: M[T_I] and M[T_S], respectively:

$$M\left[T_O\right]=\mu\left(T_O\right)+R*\sigma\left(T_O\right) \qquad (1)$$

where OPERATION is either INSERT or SELECT. The estimates are derived from the expected duration of the operation E[T_O]. It is given by the average processing time of the operation μ(T_O), and its standard deviation σ(T_O). The real value R is a confidence factor which determines together with the standard deviation σ the confidence interval or the allowable probability of the constraint violation. For example, provided Gaussian normal distribution the value R=3.0 results in 99.73% probability of not exceeding the $T_{O\,MAX}$. Data modification operations are usually not defined on streams.

B. Proposed solution

The table *D* consists of two subtables, namely *D0* and *DI* that differ in access methods. Data in *D0* is accessed by means of full scan whereas data in *DI* is indexed. All incoming data of the data stream *ds* is inserted into *D0*. *DI* contains less recent data of the stream that were moved there from *D0* during indexing schema maintenance operations in the past. The objective of the schema maintenance operation is to improve performance to meet the soft constraints $T_{I\,MAX}$ and $T_{S\,MAX}$. There are two cases that result in accomplishing the schema maintenance operation:

1) Duration of insert or select operation that is to be executed would violate $T_{I\,MAX}$ or $T_{S\,MAX}$ with high probability,
2) full scan of *D0* takes more time than access to data in *DI*.

To be able to check for these situations, some temporal statistics must be gathered during the execution of operations on ReTIn. In 1) reduction of the *DI* part may be necessary. It is done by moving the less resent data, which is considered to be obsolete to some overflow storage, or by deleting it. This data will not further be available under ReTIn constraints, but may stay in the database. The schema maintenance operation should not block and significantly delay insertion of new stream data and querying the data in *D*. We solve it such a way that the maintenance operation is performed asynchronously as a background process to insert and select operations. In addition, the maintenance operation must be atomic.

Our approach is advantageous in at least two situations. First, when the duration of a sequential scan for a select operation on *D* would take much longer than a corresponding index scanor when it would violate the constraint $T_{S\,MAX}$. Second, when updating an index would take much longer than a simple insertion of data or it would violate the constraint $T_{I\,MAX}$:

$$E\left[T_{SD0}\right]\gg E\left[T_{SDI}\right],\ E\left[T_S\right]>E\left[T_{SMAX}\right]$$
$$E\left[T_{UI}\right]\gg E\left[T_I\right],\ E\left[T_{UI}\right]>E\left[T_I MAX\right] \qquad (2)$$

where E[T_O] stands for expected duration of a corresponding operation and T_{UI} stands for index update time.

C. ReTIn Schema

The basic elements of the ReTIn indexing schema are shown in Figure 1. It consists of the hierarchy of three tables. All the tables have the same schema (*t*: timestamp, *d*: *dt*), where timestamp is an underlying DBMSs data type and *dt* is the type of an element of the data stream *ds*. Because the data type *dt* can be a composite and/or multiple-valued, the column *d* can contain arrays, subtypes or nested collections.

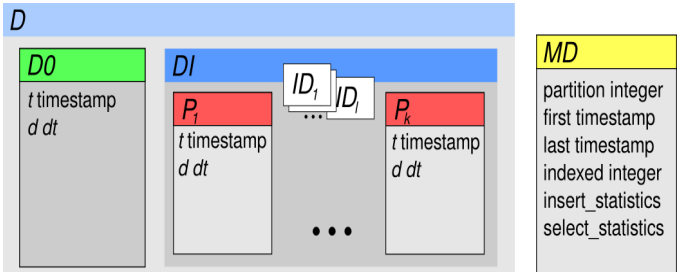


Fig. 1. Basic elements of the ReTIn indexing schema. The table *D* encapsulates subtables *D0*, which is not indexed, and indexed one *DI*. *DI* consists of partitions P_i. Metadata related to all partitions P_i and *D0* are stored in the *MD* table.

D is a virtual table that encapsulates tables *D0* and *DI*. All clients' insert and select operations run on it (encapsulation).

D0 is a base table containing the recent data of the data stream *ds* that has been inserted into *D*. The data is accessed by a full scan.

DI is a virtual table that encapsulates one or more base tables P_i (i = 1, ... k) referred to as partitions. The number of partitions k changes in time. There are one or more indexes on *DI* or each P_i.

MD is a base table that contains temporal statistics concerning the insert and select operations on tables *D*, *D0* and *DI*, as illustrated in Figure 1. It will be described in more details in the following section.

D. Operations of ReTIn

The ReTIn indexing schema provides two logical operations to its clients (operations of the DBMS):

1. INSERT INTO *D*(*d*) VALUES(*e*) – inserts an element *e* into table *D*,
2. SELECT *statement* – selects data from D.

INSERT and SELECT statements are conforming the SQL standard and they fully rely on corresponding operations of the underlying DBMS (no special operations needed). The only additional activities include logging of queries and the update of temporal statistics used for the decision whether the indexing schema maintenance operation should be performed. It is an internal operation of the ReTIn role, which changes the content of tables *D0* and *DI* in such a way that the soft constraints *TI MAX* and *TS MAX* will be met for some period of time. The maintenance is performed asynchronously to all operations.

All operations are described more formally below. Inputs, outputs (data only is considered here) and preconditions used are specified. Algorithms are described in a pseudocode.

Alg. 1. Operation INSERT

```
Input: e – a data stream element
Output:
Precondition: INSERT INTO D(d) VALUES(e) performed
INSERT INTO D0 VALUES (current timestamp, e);
update_insert_statistics();
SIGNAL "check RT constraints";
```

Algorithm 1 presents the INSERT operation. It is ensured, that the new value will always be inserted into the table *D0* without the need to update any index. The database must support (instead) triggers. We use also table inheritance (since SQL: 1999) for partitioning. The operation update_insert_statistics() updates statistics related to the insert operation. These statistics are stored in the metadata table *MD*. The operation updates sums that are necessary to compute the mean $\mu(T_I)$ and the standard deviation $\sigma(T_I)$. The statement SIGNAL represents the sending of an asynchronous message to the process responsible for the indexing schema maintenance operation. For example dbms_alert in Oracle or listen/notify concept in PostgreSQL. If the DBMS does not support this functionality, it is necessary to set a sleep period for the maintenance process, the delay can be derived from the frequency of insertions.

Alg. 2. Operation SELECT

```
Input: a select statement
Output: rs – a result set, retrieves data from D
Precondition: SELECT d FROM D … query performed
rs = EXECUTE SELECT statement;
update_query_statistics(statement);
```

Algorithm 2 presents the SELECT operation. Its execution is optimized by DBMS's query processing planner and optimizer. We suppose the optimizer uses indexes on the table *DI* (P_i) and a full scan on the table *D0* to access the data from the table *D*. Next, the update_query_statistics() operation logs the query in a log, which may be standard log of the DBMS. We use pgFouine to analyze the logs and to compute temporal statistics of of SELECT operations. For the purposes of testing, we perform some queries in the indexing_schema_maintenance() process and measure their duration. It calculates the mean $\mu(T_S)$ and standard deviation $\sigma(T_S)$ of the queries duration on *D*. Moreover, it calculates the mean value $\mu(T_{S\,DI})$ of the durations of accessing data in *DI* employing indexes and the mean value $\mu(T_{S\,D0})$ of the durations of the accessing data in *D0*.

Alg. 3. Process indexing_schema_maintenance()

```
Input: user-defined constraints, table MD
Output: schema changes
Precondition: A signal "check RT constraint"
              or a batch of INSERTs
M[T_I] = μ(T_s) + n*σ(T_s);
if M[T_I] > T_I MAX then
   raise warning "Insufficient Hardware";
M[T_S] = μ(T_s) + R*σ(T_S);
E[T_S DI] = μ(T_s DI);
E[T_S D0] = μ(T_s D0);
if M[T_S] > T_S MAX or E[T_S D0] > E[T_S DI] then {
   create new virtual table D_I';
   if M[T_S DI] > T_S then
      exclude partition P_1 from DI';
   if E[T_S D0] > E[T_S DI] then {
      data in D0 make a new partition P_k+1;
      include partition P_k+1 into DI';
   }
   create indexes for DI';
   replace DI with DI';
}
```

In algorithm 3, expressions E[*X*] and M[*X*], in accordance with (1), stand for expected value and estimated maximum value of *X*. The first condition (if) in the algorithm checks the insert operation durations to meet the soft real-time constraint $T_{I\,MAX}$. If it is violated, the situation is just reported, because the ReTIn does not use any index while inserting the data, so there is no related overhead that could be reduced.

The second condition checks the temporal constraints and defines when the indexing schema operation should be performed. Until the condition is met, the balancing of the execution time of the full scan on *D0* and the index data access on *DI* is considered to be optimal. The indexing schema maintenance operation can be executed if one or both of the following conditions are met – The duration of select operations on the indexed data part are about to break the user-defined $T_{S\,MAX}$ or sequential selects last longer than the indexed ones. In such cases the schema is changed and indexes are created. The re-indexing process is accomplished by the atomic replacement of the deprecated logical index table *DI* with *DI'*.

You can download ReTIn on PostgreSQL implementation at http://www.fit.vutbr.cz/research/prod/index.php.en?id=129 under GNU General Public License.

III. Experimental Results

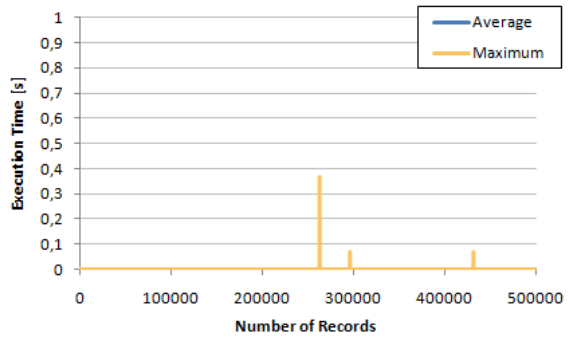
We used a dataset of meteorological observations em Global Surface Summary of Day Data (GSOD) [13] for experiments. GSOD is a product archived at the National Climatic Data Center (NCDC) to make a wide range of climatic data available to researchers and the public. The on-line data files cover the time period from 1929. They contain data from more than 9,000 stations. Each record contains the global summary of day data containing 18 surface meteorological means and maximums and other characteristics as temperature, dew point, sea level pressure, visibility, wind speed together with precipitation amount, snow depth and indicators for occurrence of fog, rain or drizzle, snow or ice pellets, hail, thunder, and tornado/funnel cloud summary. Although this is not typical data with critical real-time constrains, but their huge ammount, spatio-temporal and data stream nature and the general availability make them ideal for repeatable experiments.

The GSOD data were represented by an array of integers. Float values in the dataset were rescaled and converted into integers due to performance and memory saving reasons. Then the table D into which the data is stored in the database had schema $D(t$: timestamp, d: array of integer). ReTIn implementation based on the PostgreSQL 8.4 database management system and the Generalized Inverted Index (GIN) index, recommended for indexing of large arrays, ran on a server 2 x AMD Opteron 2435 (6 cores, 2.6GHz), 64GB RAM and 2.5TB RAID-6.

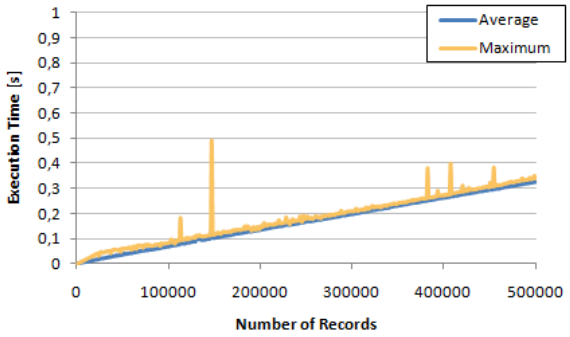
The goal of the first experiment was to show dependency of the execution times of insert and select operations on the amount of data in the database for given constraints $T_{I\,MAX}$ and $T_{S\,MAX}$. There were three approaches to access data used: unindexed data, GIN indexed data and by means of the ReTIn indexing schema. The experiment was evaluated on 500,000 records of 1950's GSOD data. Size of the table D was about 240 MB including the GIN index structure.

The methodology of the experiment was as follows: Records were sequentially inserted into the data table. Average and maximum execution times of insertions were measured for batches of 100 insertions. Average and maximum durations of queries were measured by a set of queries for batches of 1,000 insertions. The same set of queries with the *contains* array operator was used in the batches. A result set of queries contained 5 to 50% of all records in the table. Execution times were measured by stored functions on the database server. They are equivalent to the EXPLAIN ANALYZE query. During this experiment we set both $T_{I\,MAX}$ and $T_{S\,MAX}$ constraints to 0.3s and $R = 3\sigma$. The experiment was repeated 3 times to avoid random noise.

Figure 2 shows dependency of average and maximum execution times on the size of the table D without any index on data column d. This approach was very fast for insertion but execution times of queries increased linearly with the number of records. The 0.3s time constraint was permanently broken for more than 460,000 inserted records in the table. This corresponds to our expectation because of the full scan access to data.

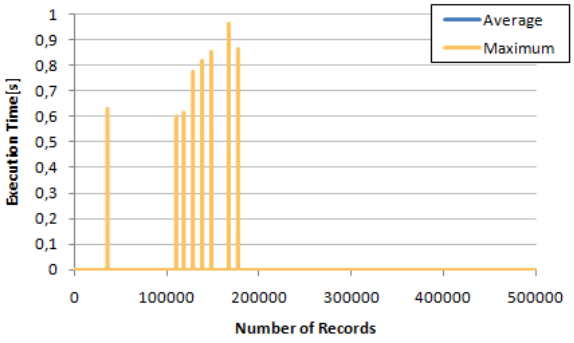


a)

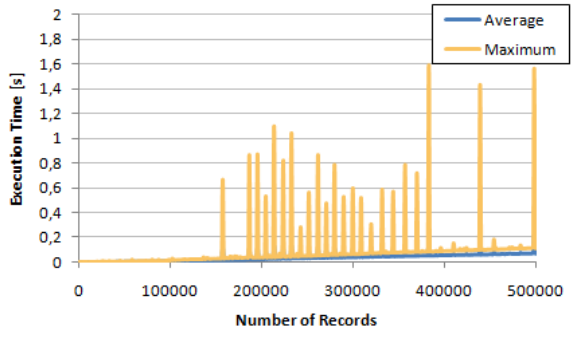


b)

Fig. 2. Execution times of (a) insertions, (b) queries on the database table without an index on column d.

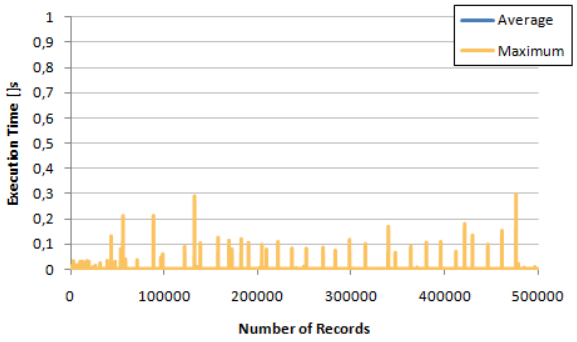


a)

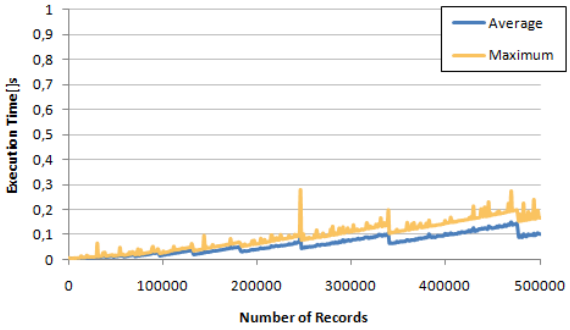


b)

Fig. 3. Execution times of (a) insertions, (b) queries on the database with the GIN index on column d.

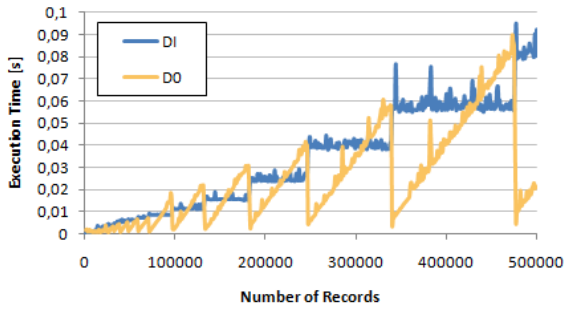


a)

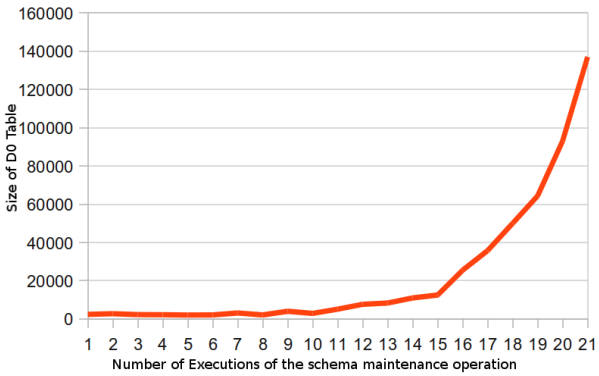


b)

Fig. 4. Execution times of (a) insertions, (b) queries on table *D* of the ReTIn index schema.



a)



b)

Fig. 5. Re-indexation aspects: (a) average times of queries on *D0* and *DI* tables, (b) dependency of the size of the *D0* table on the number of executions of the index schema maintenance operation.

Figure 3 shows the same situation as in 2, when the GIN index on data column *d* was created. The problem of this approach is shown in Figure 3 a). There are many insertion execution time peaks between 110,000 and 180,000 records. The cause of this phenomenon is the necessity to re-build the index structure. The maximum execution time of queries exceeded the value 0.3s of the $T_{S\,MAX}$ constraint many times.

The results for ReTIn are presented in Figure 4. They show the benefit of the proposed indexing schema. Maximum insertion execution times in figure 4 a), were all below the value 0.3s of the $T_{I\,MAX}$ constraint. Execution times were slower only when the indexing schema maintenance operation was performed. Execution times of queries shown in Figure 4 b) demonstrate the benefit of our approach - the ReTIn indexing schema combines a stable time of insertion with balanced query processing.

Figure 5 provides a more detailed view of the behavior of the ReTIn schema with respect to tables *D0* and *DI*. Figure 5 a) shows the decomposition of the average execution times from Figure 4 b) to the times spent by partial queries accessing data in tables *D0* and *DI*. Figure 5 b) shows the dependency of the size of the *D0* table on the number of executions of the indexing schema maintenance operations. At the beginning, the full-scan search is very fast but grows linearly. As the number of records grow, more data is searched using the index and the total execution times grows logarithmically. It has proved our hypotheses stated in section 2.

The second experiment was focused on the concurrency properties of the ReTIn. We simulated concurrent transactions by two groups of clients. The first one generated transactions containing an insert operation, the other queries. There were several threads running in parallel. We used the same set of data as in the first experiment. We evaluated the dependency of constraint violations on the number of parallel queries and insertions performed three times a second.

Table 1 shows main results of the experiment. We expected maximum violation rate about 0.3% by setting the confidence factor $R = 3\sigma$. The experiment showed that ReTIn limit for $T_{I\,MAX}$ on this hardware is about 150 transactions a second – 25 parallel insertions and 25 parallel queries 3 times a second (the row picked in bold in Table 1). This value also determines the maximum size of the data stream sliding window. For more transactions, the window size would have to be reduced to about 200,000 items for 300 transactions a second in our case. If we compare the same experiment with data stored only in table with a GIN index – see the last row in Table 1, which corresponds 150 transactions a second, we can see the benefit of ReTIn. It fails about twice less for querying and 10 times less for the data insertion.

TABLE I. THE CONCURRENCY EXPERIMENT

Threads	Queries failed	Inserts failed
	Using ReTIN indexing schema	
10+10	0.00%	0.03%
25+25	**0.52%**	**0.31%**
50+50	1.13%	0.54%
	Using simple GIN index	
25+25	**1.23%**	**3.84%**

IV. Conclusions

We have proposed, implemented and evaluated a soft real-time indexing schema called ReTIn. It makes it possible to index a portion of a data stream stored in a database effectively and to meet real-time constraints for insert and select operations with some confidence. It combines storing the most recent data unindexed and indexing less recent data. The former is advantageous from insert operation point of view, but results in a full scan access for select operations. The latter provides more effective access to the rest of data. The indexing schema maintenance operation optimizes the balance between unindexed and indexed data access with respect to the real-time constraints. It is performed asynchronously to clients' insert and select operations, which are standard SQL queries.

The experimental evaluation showed advantages in comparison with indexing all data stored in the database. We used the efficient PostgreSQL's GIN index both in our ReTIn implementation and as a competitive access method in the experiments. They showed that ReTIn behaves appropriately for insertion and selection operations on both indexed and unindexed data in the database. Moreover, it changes its behavior automatically according to the system load – it changes the width of the sliding window that defines the number of data stream elements stored in the database.

The ReTIn framework does not specify the type of index used for indexing. Current DBMSs usually provide several types, some of them are suitable for indexing complex data, similarity search etc., for example KD-tree or R-tree. Their disadvantage often is high overhead of insertions. ReTIn can cushion this problem and allow indexing data streams containing complex data, e.g. spatio-temporal and arrays.

In the future, we intent to continue experimental evaluation of ReTIn with other types of indexes. In addition, we will focus on the ReTIn deployment and optimization for our surveillance network system SUNAR and the network security project, for which it was originally designed.

Acknowledgments

This work has been supported by the research project Security-Oriented Research in Information Technology CEZ MSM0021630528, grant VG20102015006 of the Ministry of the Interior of the Czech Republic, the European Regional Development Fund in the IT4Innovations Centre of Excellence (CZ.1.05/1.1.00/02.0070) and with a financial support from the Czech Republic state budget through the Ministry of Industry and Trade.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Madison, Wisconsin, 2002, pp. 1–16.

[2] T.-W. Kuo and K.-Y. Lam, "Real-time Database Systems: An Overview of System Characteristics and Issues," in *Real-Time Database Systems*, 2001, pp. 3–8.

[3] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121–142, 2006.

[4] J. Krämer and B. Seeger, "Semantics and implementation of continuous sliding window queries over data streams," *ACM Trans. Database Syst.*, vol. 34, no. 1, pp. 1–49, 2009.

[5] M.-J. Hsieh, M.-S. Chen, and P. S. Yu, "Approximate Query Processing in Cube Streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, pp. 1557–1570, 2007.

[6] C. C. Aggarwal, *Data Streams: Models and Algorithms*. Springer US, 2007.

[7] J. Feng, Y. Wang, J. Yao, and T. Watanabe, "Multi-Granularity Aggregation Index for Data Stream," in *Cyberworlds, International Conference on*, Los Alamitos, CA, USA, 2008, pp. 767–771.

[8] G. No, S. Servigne, and R. Laurini, "The Po-tree: a Real-time Spatiotemporal Data Indexing Structure," in *Developments in Spatial Data Handling*, Springer Berlin Heidelberg, 2005, pp. 259–270.

[9] P. Chmelar, A. Lanik, and J. Mlich, "SUNAR: Surveillance Network Augmented by Retrieval," in *ACIVS 2010*, 2010, pp. 155–166.

[10] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying update streams in a soft real-time database system," *SIGMOD Rec.*, vol. 24, no. 2, pp. 245–256, 1995.

[11] T. Chiueh and L. Huang, "Efficient Real-Time Index Updates in Text Retrieval Systems," *EXPERIMENTAL COMPUTER SYSTEMS LAB, DEPARTMENT OF COMPUTER SCIENCE, STATE UNIVERSITY OF NEW*, 1999.

[12] Q. Zhu, B. Dunkel, N. Soparkar, S. Chen, B. Schiefer, and T. Lai, "A piggyback method to collect statistics for query optimization in database management systems," in *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, 1998, p. 25.

[13] "Global Surface Summary of the Day - GSOD." NOAA.