Prediction Functions in Bi-temporal Datastreams

André Bolles, Marco Grawunder, Jonas Jacobi,
Daniela Nicklas, and H.-Jürgen Appelrath

Universtät Oldenburg, Department for Computer Science

Abstract. Modern datastream management system (DSMS) assume sensor measurements to be constant valued until an update is measured. They do not consider continuously changing measurement values, although a lot of real world scenarios exist that need this essential property. For instance, modern cars use sensors, like radar, to periodically detect dynamic objects like other vehicles. The state of these objects (position and bearing) changes continuously, so that it must be predicted between two measurements. Therefore, in our work we develop a new bi-temporal stream algebra for processing continuously changing stream data. One temporal dimension covers correct order of stream elements and the other covers continuously changing measurements. Our approach guarantees deterministic query results and correct optimizability. Our implementation shows that prediction functions can be processed very efficiently.

Keywords: Datastream Management, Query Processing, Prediction.

1 Introduction and Related Work

Datastream management systems (DSMS) continuously process queries over streaming data. To prove determinism and correctness of query results and to provide optimizability, a DSMS needs a formally defined algebra. However, most existing DSMS prototypes [1–3] have no formally defined algebra. They are used to investigate processing mechanisms for datastreams, but cannot guarantee a deterministic and correct execution of continuous queries. Only PIPES [4] provides a formally defined stream algebra that guarantees deterministic results and correct behavior for query optimization. However, PIPES' relational stream algebra does not support prediction of measurements. This does not fit real world requirements. E. g. in advanced driver assistance systems (ADAS) continuously moving objects have to be tracked. To compare object detections of the last scan period with object detections of the current scan period in these systems the continuously changing states, e. g., position and bearing, have to be predicted to the same point in time. In other scenarios, such as wireless sensor networks or patient monitoring prediction of values is also useful, e. g., for energy saving or earlier hazard detection.

To support prediction functions, on the one hand specialized predictions functions must be devolped as done in [5–7]. On the other hand, these prediction functions must be used for processing streams of detected continuously changing values. On this note, especially [8] is related to our work. Here, prediction functions are attached to stream elements to reduce the communication overhead in a wireless sensor network. However, a clear integration into a query algebra that is necessary for provable correct query processing is missing. Furthermore, this work is restricted to linear or linearized prediction functions. This is not adequate for some scenarios: for example, processing dynamic models with covariance matrices leads to quadratic functions. Therefore, in our work we integrate arbitrary prediction functions into the algebra of a DSMS. For this, we use a bi-temporal model based on [9]. The first temporal dimension in our model is the stream time to cover the correct temporal processing order of elements (cf. [4]). The second dimension is the prediction time to cover continuously changing values. Our contribution is twofold:

- We formally define the semantics of our query algebra. We also show that our algebra can benefit from optimization rules.
- We present an efficient implementation of our query algebra in a DSMS.

The rest of this paper is structured as follows: We present an example scenario to illustrate prediction enabled query algebra requirements in Section 2. This algebra is formally defined in Section 3. We show the evaluation of our approach in Section 4 and give a conclusion in Section 5.

2 Stream Processing with Prediction Functions

This section illustrates the requirements and concepts for query processing over streams with prediction functions. For this, see the example query shown in Listing 1 that continuously compares the distance between the own vehicle (ego-car) and vehicles ahead to avoid a rear end collision.

```
SELECT ego.pos FROM ego RANGE 10 seconds, s2 RANGE 15 seconds
WHERE ego.speed > 30 AND s2.pos - ego.pos < 15
SET PREDFCT ego.pos:=ego.pos+ego.speed*t WHERE true
            s2.pos:=s2.pos+s2.speed*t WHERE s2.type == "vehicle"
```

Listing 1. Example Query

Measurements from two sensors are processed. Sensor ego provides measurements about the ego-car and sensor s2 detects objects ahead of the ego-car. This query compares the position of the ego-car with the positions of detected objects ahead. SET PREDFCT defines prediction functions to predict the continuously changing positions. All measurements from ego are predicted with the same prediction function while measurements from s2 are only predicted, if they represent a moving vehicle (expressed by WHERE in the SET PREDFCT clause). Figure 1 illustrates prediction function evaluation. Measurement values **a** and **b** are not compared directly, but the outcomes of their prediction functions are. So, the points in time are determined when the difference between their predicted values gets under a specified threshold (e. g., 15 meters).

The straight forward approach to process the example query (probably used in [8]) would attach matching prediction functions to each stream element and solve them for

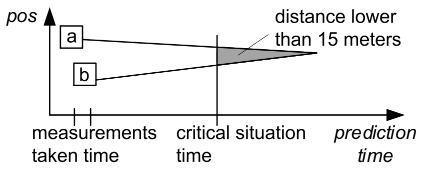

Fig. 1. Predicting specific situations

each query predicate evaluated over the element. However, in this case, multiple solving of prediction functions is necessary. To avoid this, we use prediction functions to redefine query predicates *before* query processing starts. The prediction functions `ego.pos:=ego.pos+ego.speed*t` and `s2.pos:=s2.pos+s2.speed*t` redefine the query predicate `s2.pos - ego.pos < 15` from our example query into `s2.pos+s2.speed*t - (ego.pos+ego.speed*t)< 15`. This inequality is then solved for time (denoted by variable t), which in our example leads to the following four solutions for t with mutually exclusive guards:

$$s2.pos + s2.speed * t - (ego.pos + ego.speed * t) < 15 \Leftrightarrow$$
$$\begin{cases} t < \frac{15 - s2.pos + ego.pos}{s2.speed - ego.speed} & \text{if } s2.speed - ego.speed > 0 \\ t > \frac{15 - s2.pos + ego.pos}{s2.speed - ego.speed} & \text{if } s2.speed - ego.speed < 0 \\ t & \text{if } s2.speed - ego.speed = 0 \wedge 15 - s2.pos + ego.pos > 0 \\ \emptyset & \text{if } s2.speed - ego.speed = 0 \wedge 15 - s2.pos + ego.pos \leq 0 \end{cases}$$

If there is more than one prediction function, query predicates will be redefined for each of these prediction functions. To decide which redefined predicate to use, in our solution a key is attached to each stream element that points to the corresponding prediction function.

To evaluate a redefined predicate each guard is checked for each input element. If the guard is true, the expression for t will be evaluated and used to determine the interval in which the query predicate is true. If multiple predicates are combined by `AND`/`OR`, the resulting intervals must be combined by pairwise intersection (`AND`) and union (`OR`) of the resulting intervals. Sorting the guards according to their frequency to be true can be used for performance optimization (see Section 4).

The intervals returned by redefined query predicates represent the prediction time that describes when a query predicate will be true according to the defined prediction functions. The order of stream elements and by this the windows over which the query is evaluated (cf. [4]) are not affected by these intervals. This is another temporal dimension we denote with stream time. To cover both prediction time and stream time, we use a bi-temporal model similar to [9]. In this model, the two dimensions are orthogonal. This is necessary, since stream time and prediction time do not necessarily correlate, e. g., due to network delays or system overload. If we had only one temporal dimension, predictions would cause a reorder of stream elements. Newer elements would eventually be processed before older ones and therefore deterministic results could not be guaranteed any more (e. g., a join cannot decide to prune elements). Figure 2 illustrates this 2-dimensional time domain.

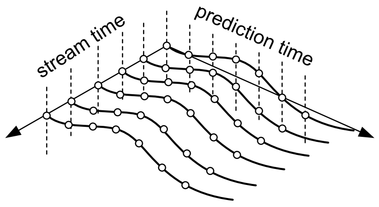

Fig. 2. Our discrete bi-temporal datastream model

Each element in the result of the query in Listing 1 describes *when* the situation expressed by the predicates will be valid. An element with stream time t_S and prediction time t_P for example tells us that we already know at time t_S that something will happen at time t_P. This information reaches the application before all elements with start timestamps (stream time) greater than t_S. Maybe, later elements (stream time) will give other information. However, this problem exists in each prediction scenario and is therefore not specific to our system.

3 Bi-temporal Stream Algebra

To guarantee deterministic results of prediction enabled queries and to take benefits of well-known relational optimization heuristics, the underlying query algebra must be formally defined. We first define our datastream model on which our algebra is based (Section 3.1) and then describe the semantics of the operators of our algebra (Section 3.2).

3.1 Datastream Model

The semantics of our operators are defined on logical datastreams similar to [4]. By this we can show the reducibility of our algebra to the well-defined relational algebra due to snapshop-reducibility.

Logical Datastream. A logical stream S^L is a set of elements $r^L = (e, t_S, t_P, i)$, where e is a relational tuple containing payload data. t_S (stream time) and t_P (prediction time) are timestamps of the finest granularity in a discrete time domain $\mathbb{T}$ [10]. The circles in Figure 2 illustrate the points in time in our bi-temporal datastream model. i is an identifier describing the measurement from which the values in e are taken. If a sensor measurement is valid for n time instants in stream time, their exist n elements in a logical stream with different values for t_S, but the same value for i.

3.2 Operators

We first define a new prediction assignment operator. Then standard operators with predicates are adapted to prediction functions. Operators with no predicates like mapping, projection and set operators can also be defined over our bitemporal datamodel. However, due to space reasons we omit details here.

Prediction Assignment. To process prediction functions, we need to assign predicted values to logical stream elements. Our prediction assignment operator ϱ is parameterized by a function f_s that decides which prediction function f_p to use for each element (`WHERE` clause in `SET PREDFCT` clause). The semantics is defined by $\varrho(S_L) = \{(\hat{e}, t_S, t_P, i) | \exists(e, t_S, t_P, i) = r \in S_L \wedge f_s(r) = f_p \wedge f_p(t_P, i) = \hat{e}\}$.

For each point in prediction time and each logical stream element this operator calculates the prediction function and sets the corresponding value for that element. Without prediction functions the values would be constant over prediction time. With prediction functions at each point in prediction time other values can be valid. We can integrate arbitrary prediction functions into ϱ.

Selection. A selection operator σ decides for each element, whether it should be in the output stream of the operator or not. Our selection operator behaves slightly different. It finds out for each element when the selection predicate p will be true according to the prediction function used for the element. So, the result of a selection predicate is not a boolean value, but a set of time intervals in which the selection predicate is true. This is shown in Figure 3. Each logical stream element is evaluated separately as defined by $\sigma(S_L, p) = \{(e, m, t_S, t_P, i) | (e, m, t_S, t_P, i) = r \in S^L \wedge p(r)\}$.

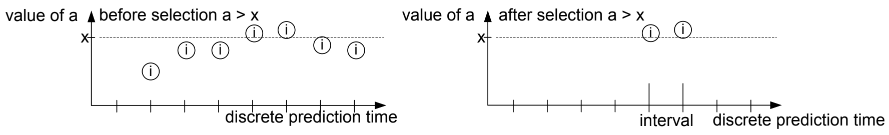

Fig. 3. Semantics of selection

Join. A join $\bowtie$ evaluates elements that are valid at the same stream time t_S. Similar to the selection predicate a join predicate p returns a set of intervals in which the join predicate is fulfilled. The formal definition of our join operator again shows that each logical element is evaluated separately: $\bowtie(S_{L,0}, S_{L,1}, p) = \{(\Lambda\!\!\!\Lambda_e(e_0, e_1), t_S, t_P, \Lambda\!\!\!\Lambda_i(i_0, i_1)) | \exists(e_k, t_S, t_P, i_k) \in S_{L,k}, k \in \{0, 1\} \wedge p((\Lambda\!\!\!\Lambda_e(e_0, e_1))\}$. The result contains the merge $\Lambda\!\!\!\Lambda_e$ of the payload data and the merge $\Lambda\!\!\!\Lambda_i$ of the identifiers.

3.3 Algebraic Optimization

Algebraic optimization allows switching operators in a query plan without changing the result of the query. Many equivalence rules, on which algebraic optimization is based, exist for the relational algebra. In this section, we show that our algebra can benefit from relational equivalence rules. For this, we define a bi-temporal snapshot at point time $t = (t_S, t_P)$ of a logical datastream by the snapshot operator τ (cf. [4, 9]): $\tau_t(S^L) := \{(e, m, i) | (e, m, t_S, t_P, i) \in S^L\}$

A snapshot of logical stream S^L is the nontemporal set of all elements being valid at the 2-dimensional point in time $t = (t_S, t_P)$. Since our operators selection, join, projection and set-operators evaluate each logical stream element separately, they are snapshopt-reducible. Thus, relational equivalence rules are valid in our algebra. Due to space limitations detailed proofs are omitted here.

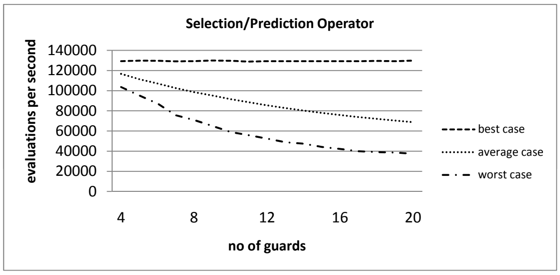

Fig. 4. Performance of Prediction Assignment and Selection

4 Evaluation

That prediction functions can increase accuracy of query results has already been shown in [5, 8, 11]. Furthermore, the results of algebraic optimization heuristics are also well-known. Thus, due to space limitations in our evaluation we concentrate on the performance of our algebraic approach. We show that our algebra is suitable for high-volume stream processing.

We implemented our algebra in our DSMS Odysseus [12] and used the predicate preprocessing step shown in Section 2. We ran our evaluation on an Intel Core 2 DUO at 2.6 GHz with 3 GB RAM. The performance of our operators depends on the guard expressions to evaluate. We used combinations of $+, -, \cdot, \div$ and $\sqrt{}$, which is a suitable set of operations for many application scenarios.

The perfomance of the selection operator is shown in Figure 4. Here, you can see the number of evaluations possible in a selection operator. By evaluation we mean the evaluation of a selection predicate (with multiple guards, see section 2). Since different prediction functions with different polynomial degrees lead to different numbers of guards to be evaluated in a selection predicate, we tested the performance of our selection operator with up to 20 guards (a quadratic function has 12 guards). Figure 4 shows that performance depends linearly on the number of guards in a selection predicate. Here you can also see three cases. The worst case means that the guard that is true is always evaluated last. With linear prediction functions (4 guards), we then have a performance of about 105000 evaluations/sec (for comparison a selection with the predicate `a.id` $\geq$ 40 can be evaluated approx. 300000 times per second). The best case means that the guard that is true is always evaluated first. In this case we have performance of about 130000 evaluations/sec. This enough in most scenarios, especially for advanced driver assistance systems where sensors usually sample at 25 Hz. Furthermore, this is for instance orders of magnitude faster than the processing in [8], due to our redefinition of predicates.

The prediction assignment operator has the same behavior as the selection operator. Here, for each possible prediction function there exists a guard that must be evaluated before the prediction function can be assigned to a stream element.

The performance of our join operator also depends linearly on the number of guards to check. However, the join operator is a little bit slower than the selection. This is, because the expressions to evaluate are larger, since they contain the prediction functions of at least two streams (or more in the case of a multiway-join). However, in the worst case and at 20 guards, we still reach a performance of 20000 evaluations/sec.

The performance can be improved by guard reorder. Here, we determine the frequency of the guards to be to be true and evaluate the most frequent guards first. Using this, we theoretically can achieve best case performance, however this depends on the stream elements.

Conclusion of Evaluation Results. The above evaluation shows that our approach is suitable for stream processing. Although not presented in this paper, using the moving objects data generator [13] we also found out that prediction functions are useful for identifying specific future traffic situations.

5 Conclusion

To the best of our knowledge, no work exists that integrates prediction functions into the query algebra of a datastream management system. Therefore, in this paper, we present a novel bi-temporal relational stream algebra that can process prediction functions. Our approach uses two dimensions of time, stream time for efficient processing of unbounded streams and prediction time for representing continuously changing measurement values. Our logical definition of datastreams and the corresponding algebra operators allow to reduce our algebra to the nontemporal relational algebra. Therefore, our algebra has the same characteristics as the relational algebra, especially optimization heuristics known from the relational algebra can be used in our algebra, too.

Our evaluation shows that a high throughput can be achieved, if prediction functions are embedded into the predicates (redefinition).

Future work concentrates on the integration of non-snapshot-reducible object tracking operators and their interaction with relational operators.

References

1. Arasu, et al.: STREAM: The Standford Stream Data Manager. IEEE Data Engineering Bulletin 26(1) (2003)
2. Abadi, et al.: The Design of the Borealis Stream Processing Engine. In: CIDR 2005 (2005)
3. Chandrasekaran, et al.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: CIDR 2003 (2003)
4. Krämer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. ACM Trans. on Database Syst. 34(1) (2009)
5. Morzy, M.: Mining frequent trajectories of moving objects for location prediction. In: Perner, P. (ed.) MLDM 2007. LNCS (LNAI), vol. 4571, pp. 667–680. Springer, Heidelberg (2007)
6. Goel, S., Imielinski, T.: Prediction-based monitoring in sensor networks: taking lessons from MPEG. SIGCOMM Comput. Commun. Rev. 31(5) (2001)
7. Lee, C.C., Chiang, Y.C., Shih, C.Y., Tsai, C.L.: Noisy time series prediction using m-estimator based robust radial basis function neural networks with growing and pruning techniques. Expert Syst. Appl. 36(3) (2009)
8. Ilarri, S., Wolfson, O., Mena, E., Illarramendi, A., Sistla, P.: A query processor for prediction-based monitoring of data streams. In: EDBT 2009. ACM, New York (2009)

9. Snodgrass, R.T., Kucera, H.: The TSQL2 Temporal Query Langauage. Kluwer Academic Publishers, Dordrecht (1995)
10. Bettini, C., Dyreson, C.E., Evans, W.S., Snodgrass, R.T., Wang, X.S.: A glossary of time granularity concepts. In: Temporal Databases, Dagstuhl (1997)
11. Tao, Y., Faloutsos, C., Papadias, D., Liu, B.: Prediction and indexing of moving objects with unknown motion patterns. In: SIGMOD 2004. ACM, New York (2004)
12. Jacobi, J., Bolles, A., Grawunder, M., Nicklas, D., Appelrath, H.J.: A physical operator algebra for prioritized elements in data streams. Computer Science - Research and Development 1(1) (2009)
13. Brinkhoff, T.: Generating network-based moving objects. In: SSDBM 2000. IEEE Computer Society, Los Alamitos (2000)