A Logic-Based Language for Data Streams

Carlo Zaniolo
Computer Science Department
UCLA
zaniolo@cs.ucla.edu

June 2012

Data Streams

- Unbounded, rapid, time-varying streams of data elements, continuous flowing on the internet and broad-band
- **Data Stream Management Systems** (DSMS) are designed to process them continuously with immediate response to new arriving tuples
- Typical applications are IS-like Most DSMS use continuous query languages that are similar to SQL or Xpath, but
- Persistent (continuous) queries on transient data, rather than transient queries on persistent data.
- Differences create many problems:
  - no **blocking queries**
  - SQL **expressive power problems are even worse.**

Outline

- Query languages for Data streams
- Why not Datalog—let us call it Streamlog?
  - Single Time-stamped Data streams
  - CWA and negation
  - Multiple streams
  - Data streams without timestamps

Time-Stamped Data Streams

A. **Input tuples enter operators in timestamp order,**
B. **Output of query operators must also be ordered.**

A stream of messages (ground facts): **msg(Time, MsgCode)**

Repeated occurrences of a "red" alarm:

$$\mathrm{repeated}(T, X) \leftarrow \mathrm{msg}(T, X),\ \mathrm{msg}(T0, X),\ T0 < T.$$

? repeated(T, red)

When 'red alarm' occurs at time **T** *event , an output tuple is produced if the red alarm had also occurred earlier, i.e. at time* **$T0 < T$.**

The Importance of Order

For repeated occurrence of code 'red' we write: **? repeated(T, red)**

This is OK: **repeated(T, X) ← msg(T, X), msg(T0, X), T > T0.**

This is not OK: **repeated(T0, X) ← msg(T, X), msg(T0, X), T > T0.**

Thus the **T0** *event comes first and then when the* **T** *event occurs, an output tuple is produced at once.*

This time warping produce out-of-order outputs. For instance (t_1 a) … (t_2 b), … (t_3 b), … (t_4 a) in the input produces (t_2 b) , (t_1 a) in the output

Negated Goals

- First occurrence of code red: $?\mathtt{first}(\mathtt{T},\mathtt{red})$

$$\mathtt{first}(\mathtt{T},\mathtt{X}) \leftarrow \quad \mathtt{msg}(\mathtt{T},\mathtt{X}), \neg\mathtt{previous}(\mathtt{T},\mathtt{X}).$$
$$\mathtt{previous}(\mathtt{T},\mathtt{X}) \leftarrow \quad \mathtt{msg}(\mathtt{T0},\mathtt{X}), \mathtt{T0} < \mathtt{T}.$$

This query uses negation on events that, according to their timestamps, are past events. The query can be answered in the present: it is non-blocking.

- Last occurrence of code red: $?\mathtt{last}(\mathtt{T},\mathtt{red})$

$$\mathtt{last}(\mathtt{T},\mathtt{Z}) \leftarrow \quad \mathtt{msg}(\mathtt{T},\mathtt{Z}), \neg\mathtt{next}(\mathtt{T},\mathtt{Z}).$$
$$\mathtt{next}(\mathtt{T},\mathtt{Z}) \leftarrow \quad \mathtt{msg}(\mathtt{T1},\mathtt{Z}), \mathtt{T1} > \mathtt{T}.$$

We do not know if the current red is the last one until we have seen the all stream. Obviously, a **blocking** query. **Thus negation can cause blocking but not always. We must understand when.**

Progressively Closed World Assumption (PCWA) for Data Streams

- PCWA for a single data stream revises the standard CWA of deductive databases with the provision that the world knowledge is expanding according to the timestamps of the arriving data stream tuples.
- CWA: Once the **p** not entailed by the given set of facts and Horn rules, then **¬p** can be safely assumed.
- PCWA: Once a **streamfact(T, . . .)** is observed in the input stream, the PCWA allows us to assume **¬streamfact(T1, . . .)** provided that **T1 < T** , and **streamfact(T1, . . .)** is not entailed by the *fact base* augmented with the stream facts having timestamp **< T.**
- Observe that we only have one stream here. Multiple streams will be discussed later.

(Strict) Sequentiality of Rules & Predicates

A Sequential rule: one positive goal has same TS as the head. The TS of the other goals are less or equal.

repeated(T, X) $\leftarrow$ msg(T, X), msg(T0, X), T0 < T.

A predicate is sequential when all the rules defining it are sequential

Strict sequentiality required for negated goals:

$$\begin{aligned} \mathtt{first}(T,X) &\leftarrow \mathtt{msg}(T,X), \neg\mathtt{previous}(T,X). \\ \mathtt{previous}(T,X) &\leftarrow \mathtt{msg}(T0,X), T0 < T. \end{aligned}$$

A strictly sequential rule: TS in the head is > than that in the goals. A predicate is strictly sequential when all the rules defining it are strictly sequential.

Stratification in Datalog

minpath(X, Y, D) ← path(X, Y, D), ¬shorter(X, Y, D).

shorter(X, Z, D) ← path(X, Z, D1), D1 < D.

path(X, Y, D) ← arc(X, Y, D).

path(X,Z,D) ← path(X,Y,D1), path(Y,Z,D2), D =D1+D2,

•Inefficient computation, since non-minimal paths are eliminated at the end of the recursive iteration, rather than as-soon-as generated.

•More general kinds of stratifications can solve this problem. E.g., XY-stratification, or Statelog, that is based on the introduction of an additional temporal argument. A big complication for the users.

•But in Streamlog the temporal argument is already there!!!!!!

Continuous shortest paths in graphs arriving as stream of arcs

$$\mathrm{path}(T, X, Z, D) \leftarrow \quad \mathrm{path}(T1, X, Y, D1), \mathrm{path}(T2, Y, Z, D2),$$
$$\mathrm{lgr}(T1, T2, T), D = D1{+}D2,$$
$$\mathrm{path}(T, X, Y, D) \leftarrow \quad \mathrm{arc}(T, X, Y, D),$$

- 2nd rule: a new arc arriving defines a new path
- 1st rule quadratic computation of transitive closure computed by joining the new delta-path tuples with the previous ones and viceversa. Thus; lgr(T1, *T2, T) means: (T=T1 and T>=T2) or (T=T2 and T>=T1)*

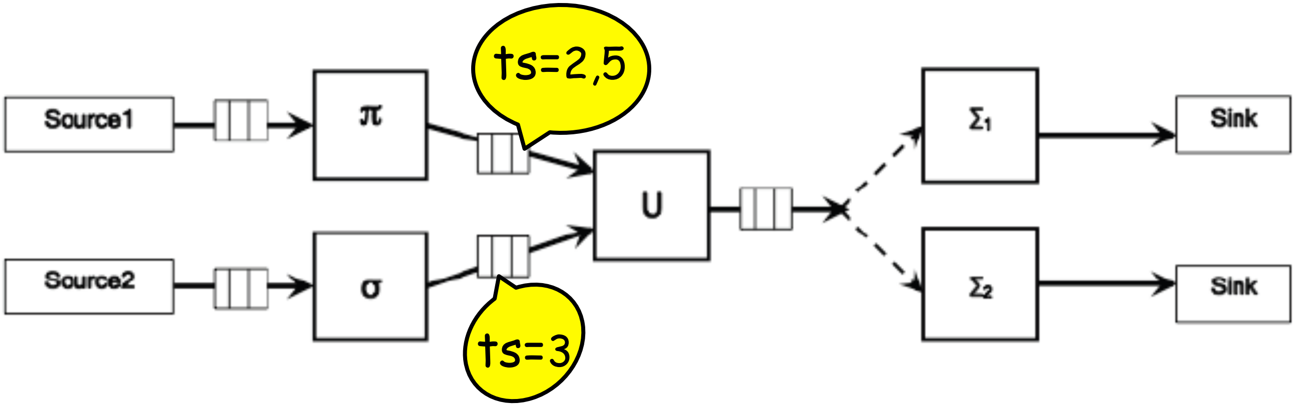
Bistate Version of a Program

1. In each rule, rename with the suffix **_new** the head predicate and the body predicates that have a timestamp equal to the that of the head,
2. Rename all the predicates in the body whose temporal argument is less than that of the head by the suffix **_old**
3. Remove the temporal arguments from the rules.

$$
\begin{array}{ll}
\mathtt{minpath\_new}(X,Y,D) \leftarrow & \mathtt{path\_new}(X,Y,D), \neg\mathtt{shorterpath\_new}(X,Y,D). \\
\mathtt{shorterpath\_new}(X,Z,D) \leftarrow & \mathtt{path\_new}(X,Z,D1), D1 \le D. \\
\mathtt{minpath\_new}(X,Z,D) \leftarrow & \mathtt{path\_new}(X,Y,D1), \mathtt{path\_new}(Y,Z,D1), \\
& \neg\mathtt{shorter\_new}(X,Z,D), D = D1 + D2. \\
\mathtt{path\_new}(X,Z,D) \leftarrow & \mathtt{path\_old}(X,Y,D1), \mathtt{path\_new}(Y,Z,D1), \\
& \neg\mathtt{shorter\_new}(X,Z,D), D = D1 + D2. \\
\mathtt{path\_new}(X,Z,D) \leftarrow & \mathtt{path\_new}(X,Y,D1), \mathtt{path\_old}(Y,Z,D1), \\
& \neg\mathtt{shorter\_new}(X,Z,D), D = D1 + D2. \\
\mathtt{path\_new}(T,X,Y,D) \leftarrow & \mathtt{arc\_new}(T,X,Y,D), \neg\mathtt{shorter\_new}(T,X,Y,D). \\
\mathtt{shorter\_new}(X,Z,D) \leftarrow & \mathtt{arc\_new}(T,\_,\_,\_), \mathtt{path\_old}(X,Z,D1), D1 \le D.
\end{array}
$$

If the bistate program is stratified then the original program is XY-stratified—and its perfect ca be computed by iterating over the computation of the bistate program.

Multiple Streams: Unions



msg(T, S) ← sensr1(T, S).

msg(T, S) ← sensr2(T, S).

Because of skew between the timestamps the above will not preserve the order.

What is the correct definition of order preserving union?

Multiple Streams and Synchronization

A. The union of two streams:

$$\mathtt{msg}(\mathrm{T1},\mathrm{S1}) \leftarrow \mathtt{sensr1}(\mathrm{T1},\mathrm{S1}).$$
$$\mathtt{msg}(\mathrm{T2},\mathrm{S2}) \leftarrow \mathtt{sensr2}(\mathrm{T2},\mathrm{S2}).$$

B. Sort-Merge oftwo streams:

$$\mathtt{msg}(\mathrm{T1},\mathrm{S1}) \leftarrow \mathtt{sensr1}(\mathrm{T1},\mathrm{S1}), \mathtt{sensr2}(\mathrm{T2},\_), \mathrm{T2} \geq \mathrm{T1}.$$
$$\mathtt{msg}(\mathrm{T2},\mathrm{S2}) \leftarrow \mathtt{sensr2}(\mathrm{T2},\mathrm{S2}), \mathtt{sensr1}(\mathrm{T1},\_), \mathrm{T1} \geq \mathrm{T2}.$$

C. Synchronized union of two streams:

$$\mathtt{msg}(\mathrm{T1},\mathrm{S1}) \leftarrow \mathtt{sensr1}(\mathrm{T1},\mathrm{S1}), \neg\mathtt{missing2}(\mathrm{T1}).$$
$$\mathtt{msg}(\mathrm{T2},\mathrm{S2}) \leftarrow \mathtt{sensr2}(\mathrm{T2},\mathrm{S2}), \neg\mathtt{missing1}(\mathrm{T2}).$$

$$\mathtt{missing2}(\mathrm{T1}) \leftarrow \mathtt{sensr2}(\mathrm{T2},\mathrm{S}), \mathrm{T2} < \mathrm{T1}.$$
$$\mathtt{missing1}(\mathrm{T2}) \leftarrow \mathtt{sensr1}(\mathrm{T1},\mathrm{S}), \mathrm{T1} < \mathrm{T2}.$$

A: how the users write it.
B: a partially blocking way in which is treated now
C: the proper characterization using negation.

Minimizing Idle Waiting in Implementation

- Generation of punctuation tuples (carrying enabling time stamps ETS) to unblock idle waiting union operators.
- At regular intervals or, on demand, via backtracking.

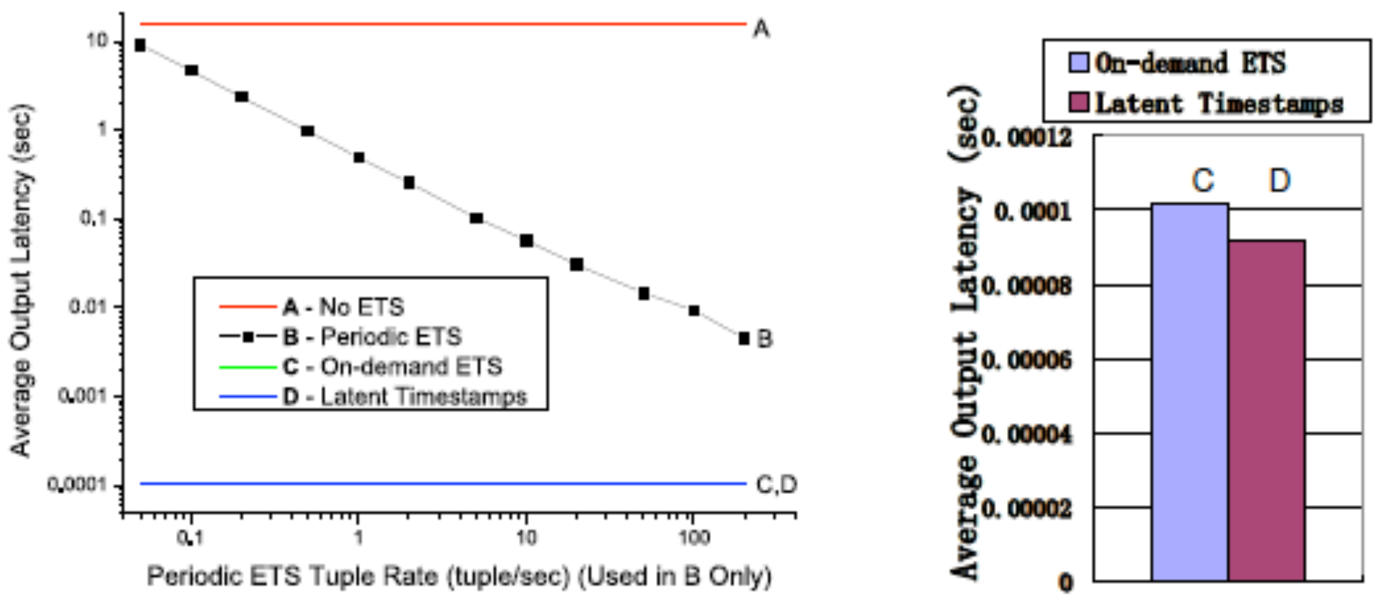


Fig. 2. Average Output Latency

Latent: same as no timestamp

Data Streams Without Timestamps

$$\mathtt{singlearc}(\mathtt{Time},\mathtt{From},\mathtt{To},\mathtt{Cost}) \leftarrow \quad \mathtt{darc}(\mathtt{From},\mathtt{To},\mathtt{Cost}),\mathtt{uts}(\mathtt{Time}).$$

uts(Time) *generates a unique timestamp every time it is called*

Iteration-free, incremental computation of transitive closure

$$\begin{aligned}
\mathtt{tc}(\mathtt{T},\mathtt{X},\mathtt{Y}) &\leftarrow \quad \mathtt{sarc}(\mathtt{T},\mathtt{X},\mathtt{Y}).\\
\mathtt{tc}(\mathtt{T},\mathtt{X},\mathtt{Z}) &\leftarrow \quad \mathtt{sarc}(\mathtt{T},\mathtt{X},\mathtt{Y}),\mathtt{tc}(\mathtt{T1},\mathtt{Y},\mathtt{Z}),\mathtt{T1} < \mathtt{T}.\\
\mathtt{tc}(\mathtt{T},\mathtt{X},\mathtt{Z}) &\leftarrow \quad \mathtt{tc}(\mathtt{T1},\mathtt{X},\mathtt{Y}),\mathtt{sarc}(\mathtt{T},\mathtt{Y},\mathtt{Z}),\mathtt{T1} < \mathtt{T}.\\
\mathtt{tc}(\mathtt{T},\mathtt{X},\mathtt{Z}) &\leftarrow \quad \mathtt{tc}(\mathtt{T1},\mathtt{X},\mathtt{Y}),\mathtt{sarc}(\mathtt{T},\mathtt{Y},\mathtt{W}),\mathtt{tc}(\mathtt{T2},\mathtt{W},\mathtt{Z}),\mathtt{T1} < \mathtt{T},\mathtt{T2} < \mathtt{T}.
\end{aligned}$$

These rules are inspired by the active rules used in concrete view maintenance.

- Negated goals to eliminate shorter paths should also be used here too
- Aggregates can be easily defined using uts(_).
- Transitive closure could also be defined as an aggregate.

Conclusion

- Non-monotonic reasoning for data streams can be supported quite naturally and efficiently using simple extensions of Datalog.
- We introduced rigorous logical foundations for continuous query languages (that were sorely lacking formal foundations)
- These are practical solutions that significantly enhance the expressive power of continuous query languages.
- Future directions: a unified language for stored data and data streams, and interoperability with query continuous query languages of data streams.
- Much work remains to be done …

Thank you!

References

1. B. Babcock, S. Babu, M. Datar, R. Motawani, and J. Widom. Models and issues in data stream systems.In PODS, 2002.
2. Yijian Bai, Hetal Thakkar, Haixun Wang, Chang Luo, and Carlo Zaniolo. A data stream language andsystem designed for power and extensibility. In CIKM, 2006
3. Yijian Bai, Hetal Thakkar, Haixun Wang, and Carlo Zaniolo. Timestamp management and query execution models in data stream management systems. IEEE Internet Computing, 12(6):13{21, 2008.
4. Yuri Gurevich, Dirk Leinders, and Jan Van den Bussche. A theory of stream queriesDatabase Programming Languages. DBPL 2007.
5. Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Data models and query language for data streams. In VLDB 2004.
6. Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. From regular expressions to nested words: Unifying languages and query execution forrelational and xml sequences. In VLDB 2010.
7. P.Tucker, D. Maier, and T.Sheard. Applying punctuation schemes to queries over continuous data streams. IEEE Data Engineering Bulletin,26(1):33{40, 2003.
8. Arcot Rajasekar, Jorge Lobo, Jack Minker. Weak generalized closed world assumption. J. Autom. Reasoning, 5(3), 1989.
9. Raymond Reiter. Deductive question-answering on relational data bases. In Herve Gallaire and Jack Minker, editors, Logic and Data Bases, Symposium on Logic and Data Bases, Toulouse, 1977.
10. Hetal Thakkar, Nikolay Laptev, Hamid Mousavi,Barzan Mozafari, Vincenzo Russo, and Carlo Zaniolo.Smm: a data stream management system for knowledge discovery. In ICDE, page 1, 2011.