Streaming Data

Susan B. Davidson

CIS 700: Advanced Topics in Databases

MW 1:30-3

Towne 309

http://www.cis.upenn.edu/~susan/cis700/homepage.html

Streaming Data

- Many modern applications require long-running, *continuous* queries over unbounded streams of data
  - Network monitoring
  - Financial analysis
  - Manufacturing
  - Sensor networks
- Contrast this with the traditional database setting of *one-time* queries over finite stored data sets.

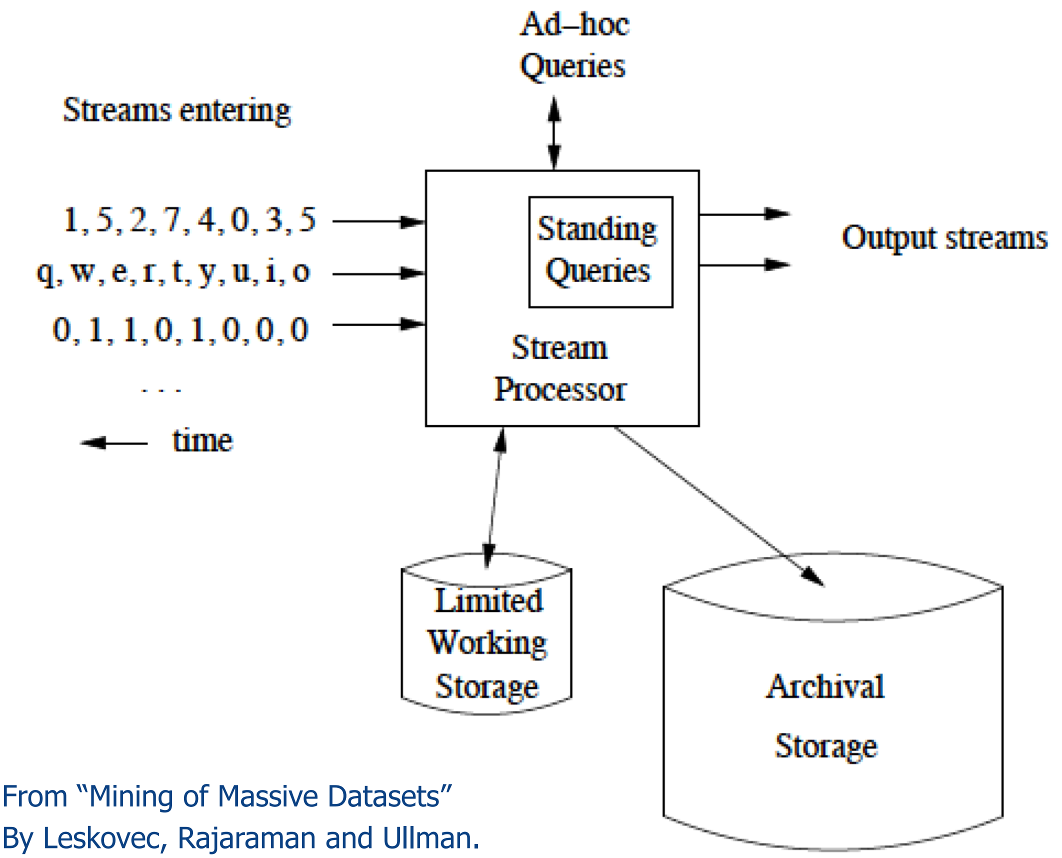
Example: oceanography

- Suppose we are collecting ocean surface temperature/ surface height data using floating sensors
  - Millions of sensors each sending back a stream at the rate of 10 readings per second
  - This could easily become several terabtyes of data per day, cannot be kept in working storage.
- Sample continuous ("standing") queries
  - Output an alert whenever the temperature exceeds 25 degrees centigrade
  - Produce the average of the 24 most recent readings
  - Produce the highest temperature recorded, or average temperature over all recordings

Example: Web sites

- Web sites receives streams of various types
  - Google receives several hundred million search queries per day
  - Yahoo! accepts billions of “clicks” per day
- Many interesting things can be learned
  - An increase in queries like “sore throat” signals the spread of viruses.
  - A sudden increase in the click rate for a link could indicate some news connected to that page, or it could mean that the link is broken and needs to be repaired.
- One approach to handle ad-hoc queries is to store a sliding window of each stream
  - All inputs in last t time units
  - Last k inputs

Data-stream management system architecture



From “Mining of Massive Datasets”
By Leskovec, Rajaraman and Ullman.

Issues in stream processing

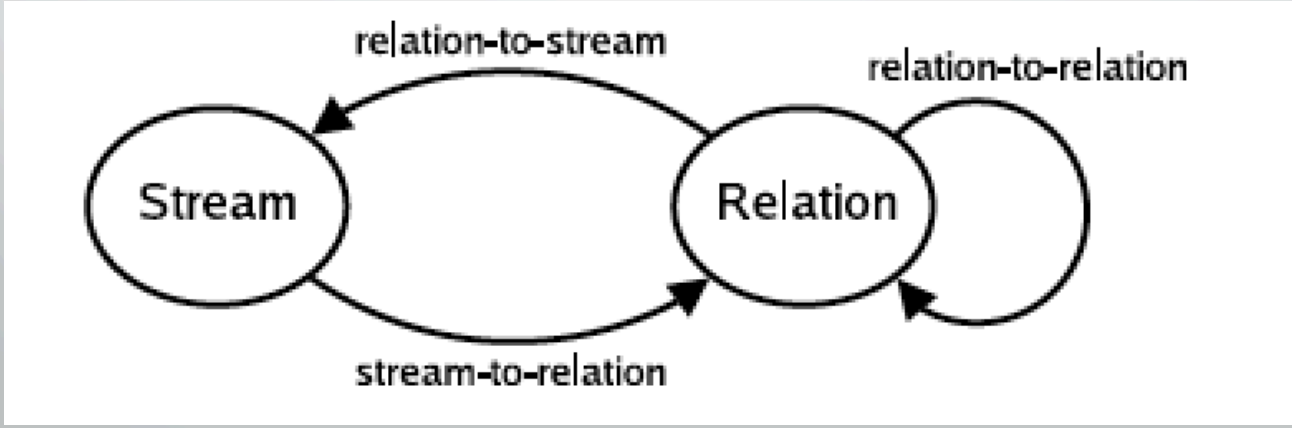
- Streams deliver elements rapidly, and elements must be processed in real time
  - Algorithms should be executed in main memory
- There may be many streams
  - Even if each stream can easily be executed in main memory, the combination may exceed available memory
  - Common techniques: approximation, hashing
- For complex, ad-hoc queries: store a sliding window of each stream in the working store.
  - Most recent n elements of a stream, for some n
  - All the elements that arrived within the last t time units

Several directions have been taken…

- Database management systems perspective
  - Stanford STREAM project
  - NiagaraCQ (Wisconsin and OGI)
- Algorithmic perspective
  - Approximation and hashing techniques are commonly explored.
- Data stream processing engines (e.g. Discretized stream, Drizzle, Flink)

STREAM: the Stanford project

- Two data types:
  - Stream: unbounded bag of pairs (s,t) where s is a tuple and t is a timestamp
  - Relation: time-varying bag of tuples, R(t) denotes an instantaneous relation



CQL: Stream-to-relation operators

- Stream-to-relation operators are based on sliding windows:
  - *Tuple-based*: R(t) contains the N tuples of stream S with the largest timestamps <=t
  - *Time-based*: R(t) contains all tuples of S with timestamps between t-w and t.
  - *Partitioned sliding window*: partitions S into different substreams based on equality of attributes A1,…, Ak and computes a tuple-based sliding window of size N independently on each substream, then take the union of the windows to produce the output relation

CQL: Relation-to-stream operators

- *Istream*: contains (s, t) whenever tuple s is inserted into R at time t
- *Dstream*: contains (s, t) whenever tuple s is deleted from R at time t
- *Rstream*: contains (s, t) whenever tuple s is in R(t), i.e. every current tuple of R is streamed at every time instant

Select Istream(*) From S [Rows unbounded] Where S.A>10

or more intuitively:

Select * From S Where S.A>10

CQL, more examples

Select * From S1 [Rows 1000], S2 [Range 2 minutes]
Where S1.A=S2.A and S1.A>10

Select Istream(S1.A) From S1 [Rows 1000], S2 [Range 2 minutes]
Where S1.A=S2.A and S1.A>10

Select Rstream(S.A, R.B) From S [Now], R
Where S.A=R.A

Query plans for continuous queries

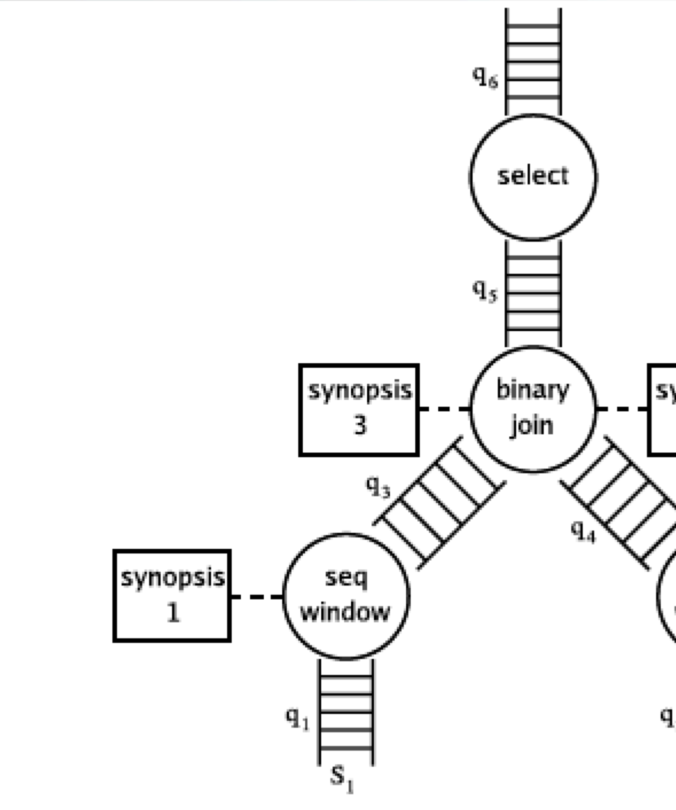
- Query plans consist of
  - **Operators**, which perform the actual processing
  - **Queues**, which buffer tuples as they move between operators
  - **Synopses**, which store operator state
- Each operator reads from one or more input queues, processes the input, and writes output to an output queue.
- All queues enforce nondecreasing timestamps.
- Synopses store state that may be required for future evaluation of an operator, and are shared between operators whenever possible.
  - E.g. materialize the contents of a sliding relation or the result of a subquery

Operators in CQL query plans

Name	Operator Type	Description
`select`	relation-to-relation	Filters elements based on predicate(s)
`project`	relation-to-relation	Duplicate-preserving projection
`binary-join`	relation-to-relation	Joins two input relations
`mjoin`	relation-to-relation	Multiway join from [22]
`union`	relation-to-relation	Bag union
`except`	relation-to-relation	Bag difference
`intersect`	relation-to-relation	Bag intersection
`antisemijoin`	relation-to-relation	Antisemijoin of two input relations
`aggregate`	relation-to-relation	Performs grouping and aggregation
`duplicate-eliminate`	relation-to-relation	Performs duplicate elimination
`seq-window`	stream-to-relation	Implements time-based, tuple-based, and partitioned windows
`i-stream`	relation-to-stream	Implements *Istream* semantics
`d-stream`	relation-to-stream	Implements *Dstream* semantics
`r-stream`	relation-to-stream	Implements *Rstream* semantics

Example

Select * from S1 [Rows 1000], S2 [Range 2 minutes]
Where S1.A=S2.A and S1.A>10



Tuple-based windows do not commute with filter conditions.

Performance in a time-varying landscape

- Novel optimizations
  - Synopsis sharing
  - Constraints on streams
  - Operator scheduling
- Monitoring and adaptive query processing
  - *Profiler* collects and maintains statistics about stream and plan characteristics, e.g. constraints
  - *Reoptimizer* ensures that plans and memory structures are efficient for current characteristics, e.g. join orders, adding/deleting subresult caches
- Approximation

Optimization: synopsis sharing

- Use “stubs” to index into shared synopses
  - Within a query, e.q. Synopsis 1 and Synopsis 3
  - Across queries, e.g.

Select * from S1 [Rows 1000], S2 [Range 2 minutes]
Where S1.A=S2.A and S1.A>10

Select A, Max(B) From S1 [Rows 200] Group By A

Optimizations: constraints

- *Referential integrity k* on a many-one join: bound *k* on the delay between the arrival of a tuple on the "many" stream and the arrival of its joining "one" tuple on the other stream.
- *Ordered-arrival k-constraint* on a stream attribute A: bound *k* on the amount of reordering in values of A.
  - For any tuple *s* in stream *S*, for all tuples *s'* that arrive at least *k* + 1 elements after *s*, it must be true that *s':A>= s:A*.
- *Clustered-arrival k-constraint* on a stream attribute A: bound *k* on the distance between any two elements that have the same value of A.

Approximation

- Data streams may be bursty with peaks during which system resources are exhausted.
  - *CPU-limited*: data arrival rate is so high that there is insufficient CPU time to process each stream element
  - *Memory-limited*: total state required for all queries may exceed available memory
- **Solution:** degrade accuracy by providing approximate answers during load spikes
  - *CPU-limited*: drop elements before they are processed
  - *Memory-limited*: selectively retain some state and discard the rest

CPU-limited computation

- Introduce sampling operators that probabilistically drop stream elements as they are input to the query plan.
- For example, suppose there is set of sliding window aggregation queries
  - **Goal**: sample the inputs to minimize the maximum relative error across all queries, i.e. keep the relative error the same for all queries
  - **Assume:** for each Qi, know the mean and standard deviation of input stream values as well as the window size (can be collected by the profiler)
  - Then can use the Hoeffding inequality to derive a bound on the probability that the relative error exceeds a given threshold for a given sampling rate.

```
Select avg(temp) From SensorReadings [Range 5 minutes]
```

Memory-limited computation

- Several optimizations are aimed at minimizing memory devoted to queues and synopsis sizes (e.g. synopsis sharing, operator scheduling, using constraints), but memory may still be a limitation
  - Spilling to disk may not be feasible as it is too slow
- Focus on reducing synopsis
  - Introducing a new window or shrinking an existing window
  - Maintaining a sample of the intended synopsis content
  - Using histograms or wavelets when the synopsis is used for aggregation
  - Using Bloom filters for duplicate elimination, set difference or set intersection
  - Lowering k-values for known k-constraints

Conclusions

- Many modern applications require continuous queries over streaming data
- Cannot directly apply relational semantics, need to introduce stream→ relation and relation→ stream operators
- Optimizing continuous queries requires a new set of tricks
  - Sharing state and computation within and across query plans
  - Using inferred constraints on data streams
  - Adaptive query processing
  - Load-shedding and approximations