DATA STREAMS AND DATABASES

CS121: Introduction to Relational Database Systems

Fall 2014 – Lecture 22

Static and Dynamic Data Sets

- So far, have discussed relatively static databases
  - Data may change slowly over time…
  - Queries and updates operate against a static data-set
  - Especially true in context of transactions: within each txn, database appears to be unchanged by other txns
- Increasingly common to have <u>data streams</u> generated by various sources
  - An infinite, time-ordered sequence of tuples
  - Tuples have a particular schema, as before
  - One attribute in the tuple schema is a timestamp τ

Data Streams: Examples

- Many different examples of data streams

- Stock market data!
- Example: Volume-Weighted Average Price (VWAP)
  - Value computed over a time-window of stock trades
  - Window is fixed size, contains *n* stock trades
  - Trade *i* has price P_i, with Q_i shares changing hands
  - $P_{VWAP} = \Sigma P_i Q_i \div \Sigma Q_i$

Data Streams: Examples (2)

- Traffic-flow sensor networks generate data streams
  - Various estimates of traffic-flow characteristics, e.g. time/ space mean speed, density, flow
- Seismographic networks generate many data streams!
  - Individual seismometers generate multiple data streams along different axes (vertical, N/S, E/W)
  - Other software consumes seismic data streams, identifies earthquakes, produces other data streams
- Computer network security monitoring systems
- Sensor networks in plants, factories, etc.

Data Streams in Relational Databases?

- How to implement Volume-Weighted Average Price (VWAP) computation with a relational database?
- Need to store the stream of stock prices into a table
  - Each stock price must have a timestamp
  - Example schema:
    - *stock_sales*(*sale_time*, *ticker_symbol*, *num_shares*, *price_per_share*)
- As new stock sales occur, store them in the table
- Need to eventually remove records from this table when we don't care anymore

Data Streams in Relational Databases?

- Periodically compute the VWAP against this table
  - ```
    SELECT ticker_symbol,
           SUM(price_per_share * num_shares) /
           SUM(num_shares)
    FROM stock_sales
    WHERE sale_time >= NOW() - INTERVAL 5 MINUTE
    GROUP BY ticker_symbol;
    ```
  - An index on (*sale_time*) will help select the rows in the window of interest
- Issues?
  - Performance: we throw away a significant amount of work every time the query is recomputed!
    ```

Data Streams in Relational Databases?

- Our query:
  - ```
    SELECT ticker_symbol,
           SUM(price_per_share * num_shares) /
           SUM(num_shares)
    FROM stock_sales
    WHERE sale_time >= NOW() - INTERVAL 5 MINUTE
    GROUP BY ticker_symbol;
    ```
  - *Can we do this incrementally instead?*
- Can easily apply same concepts as with materialized views to save and update intermediate state
  - As rows enter and leave the window of interest, we can update our rolling averages for each stock
  - Should make it very fast to generate the desired results!
    ```

Data Streams and Queries

- What if we want to update the output of our query every time the input data stream changes?
  - e.g. every time more stock values arrive, we update the corresponding VWAP immediately
  - (e.g. could implement this with triggers on *stock_sales*)
- What if we want to join the data stream against one or more relations?
- What if we want to generate a data stream based on the changes made to a relation?

Data Stream Management Systems

- Over the last few decades, many efforts to build Data Stream Management Systems (DSMS)
  - Like DataBase Management Systems (DBMS), but able to handle data streams as well as static tables
- Also called Complex Event Processing (CEP) systems
- Several approaches to building these systems…
- A popular approach:
  - Extend the relational model to add stream processing capabilities
  - (i.e. reuse the existing work of relational databases!)

Data Stream Management Systems (2)

- Many major research projects on relation-oriented stream databases:
  - Aurora/Borealis (Brown, Brandeis, MIT)
  - STREAM (Stanford)
  - TelegraphCQ (Berkeley)
- Commercial stream databases:
  - StreamBase (commercial version of Aurora/Borealis)
  - Truviso (commercial version of TelegraphCQ; acquired May 2012 by Cisco)
  - TIBCO Business Events, Oracle BAM

Stream Database Implementations

- Some stream databases were built by extending existing relational databases
  - e.g. TelegraphCQ and Truviso are extensions of PostgreSQL that incorporate data streams
  - Existing SQL syntax is extended to handle stream declarations, windowed queries on streams, etc.
- Many others are built from scratch
  - Custom stream-processing engines that are database-agnostic, or that don't use a relational database
  - Usually have a specific focus, e.g. high-volume data streams, low-latency results, specific kinds of queries, etc.
- Today: focus on relation-oriented stream databases

Data Stream Management Systems (3)

- In a relational database:
  - Data is static! (As long as no DML is issued…)
  - Issue queries against DB whenever we need results.
- In a stream database:
  - Stream data changes continually! Thus, results of queries against a stream also change continually.
  - Such queries are called <u>continuous queries</u>.
  - Register continuous queries with the database server.
  - As stream data changes, DB can incrementally update and output query results efficiently.

Stream Data Model

- A data stream S is an infinite, time-ordered multiset of tuples, one attribute being a timestamp τ
  - τ denotes the logical arrival time of the tuple into the system
- A relation R is an unordered multiset of tuples that varies over time
  - $R(\tau)$ is the version of the relation at a point in time τ
- A continuous query Q is constructed from a tree of operators against streams S_i and relations R_i

Continuous Query Operators

- Relation-to-relation operators take one or two relations and produce a relation
  - Exactly like standard relational algebra, except that relations in stream databases have a notion of time
  - Uses most recent versions of involved relations $R_i(\tau)$
- Relation-to-stream operators take a relation and produce a data stream
  - Typically defined to produce a stream containing changes made to a relation R_i
  - e.g. ISTREAM(R) produces tuples inserted into R at time τ
    - A stream of tuples $\langle s, \tau \rangle$ where $s \in R(\tau) - R(\tau - 1)$
  - e.g. RSTREAM(R) produces a stream of all tuples in R at τ

Continuous Query Operators (2)

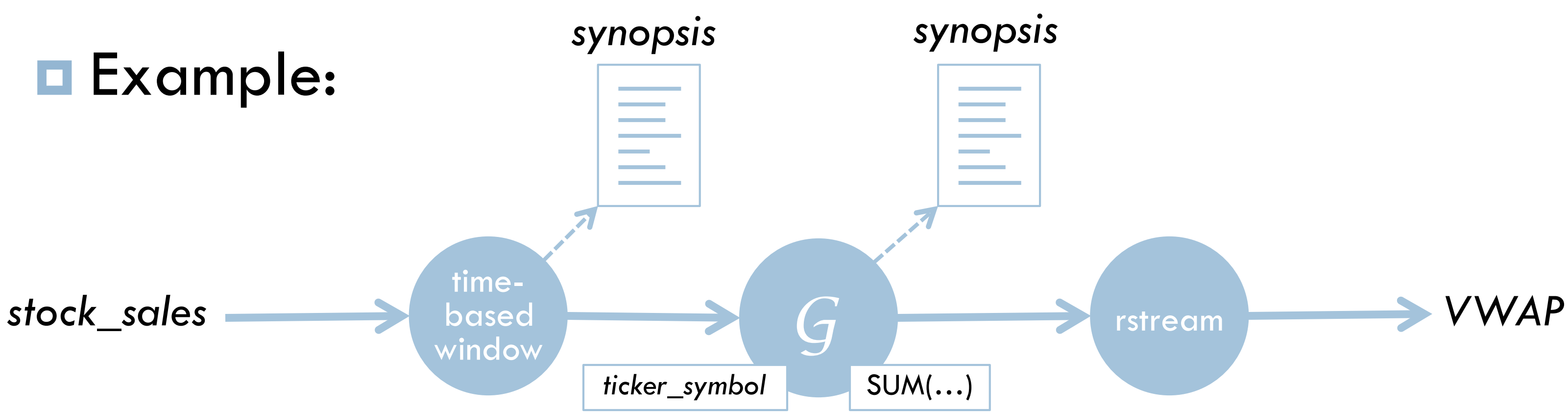
- Stream-to-relation operators take a data stream *S* and produce a relation *R*
  - Most continuous queries only care about the *recent* tuples in a data stream…
  - Define a sliding window on a stream that ends at time τ
- Tuple-based sliding windows contain the *N* most recent tuples from the data stream *S*
- Time-based sliding windows contain tuples from *S* that fall in a range of timestamps
  - $R(\tau)$ contains all tuples from *S* with timestamp in $[\tau - \omega, \tau]$
  - Also provide support for "now" tuples, where $\omega = 0$

Continuous Queries

- When a continuous query is added to the database, a plan is constructed from these operators
- Example: compute volume-weighted average stock prices from stream of stock sales
  - Convert stock-sale data stream into a relation using a time-based sliding-window operator over last 5 minutes of data
  - Compute a relation containing aggregates using standard grouping/aggregation operations
  - Convert entire relation into another stream containing results
- Problem: as stated, this is still expensive
  - Every time, results are computed entirely from scratch!

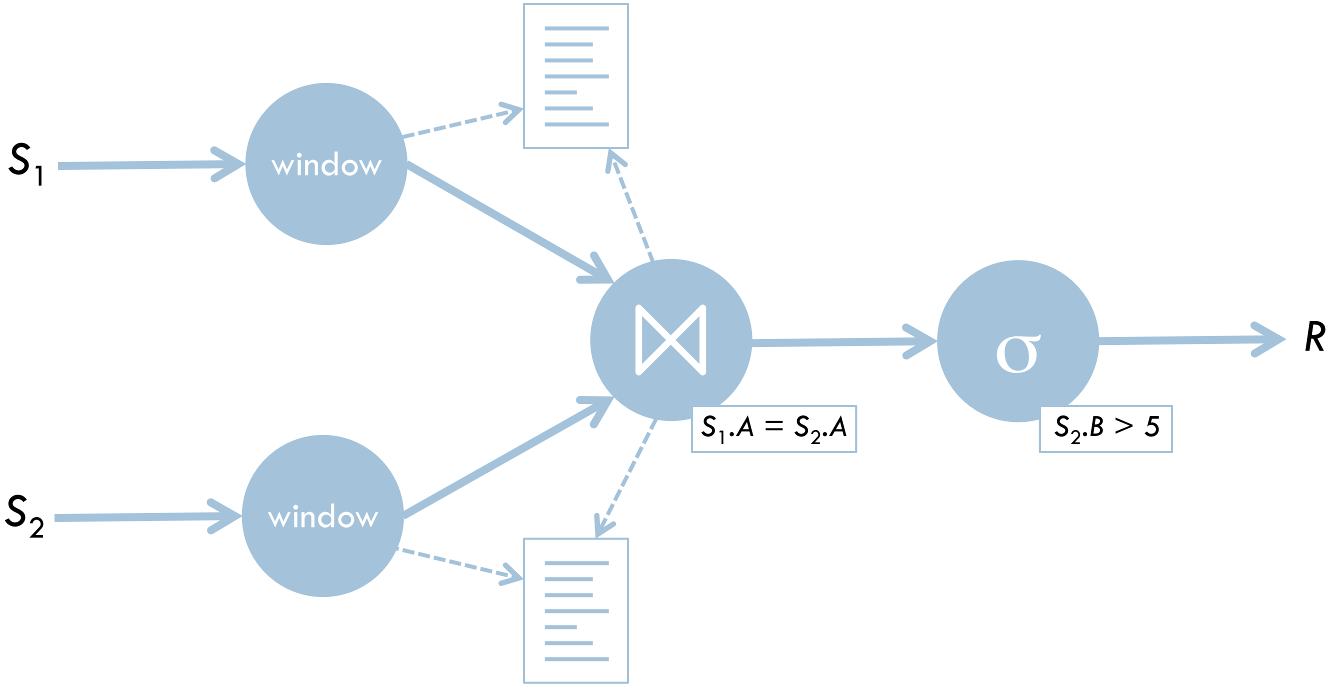
Continuous Queries (2)

- A continuous query operator can maintain a synopsis of its most recent results
  - A relation containing the rows used to generate results for most recent time τ
  - When rows enter or exit the sliding window, synopsis can be incrementally updated very efficiently
  - Example:



Continuous Queries (3)

- Can construct very complex queries, even with these [relatively] simple operators
- Example: windowed join of two data streams
  - Shared synopses across different nodes



Continuous Query Languages

- Some continuous query languages (CQL) are extensions of SQL
  - Particularly in implementations built on relational DBs
- Include ability to create and remove streams
  - ```
    CREATE STREAM stock_sales (
        sale_time TIMESTAMP,
        ticker_symbol VARCHAR(10),
        num_shares INTEGER,
        price_per_share NUMERIC(7, 2)
    );
    ```
  - `DROP STREAM stock_sales;`
- (Actual stream data usually arrives over network and must be in a specific format for the database to use it)
    ```

Continuous Query Languages (2)

- Need a way to indicate what column is timestamp τ
- Example: Truviso syntax:
  - ```
    CREATE STREAM stock_sales (
            sale_time TIMESTAMP CQTIME
                USER GENERATED,
            ticker_symbol VARCHAR(10),
            num_shares INTEGER,
            price_per_share NUMERIC(7, 2)
    );
    ```
  - Can also have system-generated timestamps
- Other stream databases use similar mechanisms, e.g. TelegraphCQ has **`TIMESTAMPCOLUMN`** modifier
    ```

Continuous Query Languages (3)

- Can also specify if a stream is archived or not
  - An archived stream can be queried for historical data; an unarchived stream cannot.
  - Most CQLs have **`TYPE [ARCHIVED|UNARCHIVED]`**
- To preserve historical data:
  - ```
    CREATE STREAM stock_sales (
        sale_time TIMESTAMP CQTIME
            USER GENERATED,
        ticker_symbol VARCHAR(10),
        num_shares INTEGER,
        price_per_share NUMERIC(7, 2)
    ) TYPE ARCHIVED;
    ```
    ```

Continuous Queries

- Can issue queries against both relations and streams
- If a stream is involved, need to specify the window!
- Stanford STREAM CQL annotates streams within the query:
  - ```
    SELECT ticker_symbol,
           SUM(num_shares * price_per_share) /
           SUM(num_shares)
    FROM stock_sales [RANGE 5 MINUTES]
    GROUP BY ticker_symbol;
    ```
- Truviso has a similar annotation:
  - ```
    SELECT ticker_symbol,
           SUM(num_shares * price_per_share) /
           SUM(num_shares)
    FROM stock_sales < VISIBLE '5 MINUTES' >
    GROUP BY ticker_symbol;
    ```

Continuous Queries (2)

- TelegraphCQ has a separate window specification:
  - ```
    SELECT ticker_symbol,
           SUM(num_shares * price_per_share) /
           SUM(num_shares)
    FROM stock_sales
    GROUP BY ticker_symbol
    WINDOW stock_sales ['5 MINUTES'];
    ```
    ```

Derived Streams

- Continuous queries run once and produce their results, just like standard queries
- Can create <u>derived streams</u> from continuous queries
  - Query is persistent, and produces a data stream of results
  - Results in output data stream have timestamps, as expected
- Truviso syntax:
  - ```
    CREATE STREAM vol_weighted_avg_price AS
        (SELECT ticker_symbol,
                SUM(num_shares * price_per_share) /
                SUM(num_shares)
         FROM stock_sales < VISIBLE '5 MINUTES'
                            ADVANCE '5 SECONDS' >
         GROUP BY ticker_symbol);
    ```
  - **ADVANCE** keyword specifies how often to generate results
    ```

Stream-Processing Challenges

- Many challenges in implementing stream databases
- Frequently, data streams are bursty and can have very high volumes
  - Peak message rate >> average message rate
  - Latency between input data and results skyrockets
- Most common approach: load-shedding
  - Simply drop some of the input tuples out of the stream!
  - Stream DBs use statistics gathered about stream data to drop/summarize tuples to maximize accuracy of results
  - May involve dropping tuples from multiple sources in a query to ensure accuracy is maximized

Stream-Processing Challenges (2)

- Other databases use windowed/aggregate operators to achieve a similar result:
  - Use windowed aggregate operations to generate multiple granularities of a given data stream
  - Different granularities will produce different volumes of tuples…
- As system load varies, can react very easily by changing the granularity of data being used
  - Choose the finest granularity of input data that the current system load will allow

Stream-Processing Challenges (3)

- Tuples in a stream don't always arrive in order!
  - A tuple's timestamp can be set by database when the tuple arrives, or it can be an externally specified field
  - e.g. stock-sale records might already include timestamp
- Generally a characteristic of a specific stream…
  - Can specify a stream's slack: the maximum "out-of-order"-ness that will be allowed
  - Tuples that arrive later than the specified slack are ignored!

Stream-Processing Challenges (4)

- Truviso example:
  - ```
    CREATE STREAM stock_sales (
        sale_time TIMESTAMP CQTIME
            USER GENERATED SLACK '1 MINUTE',
        ticker_symbol VARCHAR(10),
        num_shares INTEGER,
        price_per_share NUMERIC(7, 2)
    );
    ```
- Problem: if tuples can arrive out of order, a query may have already generated invalid results…
  - Could simply delay outputting results by the specified amount of slack. Then we know results won't change…
    ```

Stream-Processing Challenges (5)

- Another approach: output <u>revisions</u> to answers!
- Some input data streams can also include revisions to previous records
  - (commercial stock ticker feeds, for example)
  - e.g. correct invalid values, add extra rows, remove rows
- A few stream databases (e.g. Borealis) can handle revisions on data streams
- If all tuples affected by revision are still in memory:
  - Recompute affected results incrementally, using old and new versions of dataset for the affected timestamp(s)
  - Only output new records that actually changed
  - (Revised outputs may force other revisions to be made…)

Stream-Processing Challenges (6)

- If tuples affected by revision no longer in memory:
  - Must scan and replay archived data-stream tuples to find all relevant tuples
- Also likely that query nodes' synopses will no longer contain data for the affected timestamps ☹
  - Must recompute all work from scratch; can't use incremental updates
- As before, only issue revised output records for results that actually change
- If revisions are far enough in past, may cause *many* results to be recomputed and revised ☹

DSMS Summary

- Possible to use relational databases to continuously process data streams, but not very efficient!
- Relation-oriented data stream management systems (DSMS) blend together standard relational model databases with stream-processing capabilities
- Very powerful solution for problems involving data stream processing!

References

- STREAM: The Stanford Data Stream Management System [Arasu et al]
  - Introduction to how the relational model is extended to include data stream processing
- Models and Issues in Data Stream Systems [Babcock et al]
  - Discussion of data models and query languages used in relational data stream management systems
- The 8 Requirements of Real-Time Stream Processing [Stonebraker, Çetintemel, Zdonik]
  - A good summary of the major requirements of DSMSes
- Revision Processing in a Stream Processing Engine [Ryvkina, Maskey, Cherniack, Zdonik]
  - Description of Borealis revision processing

References (2)

- Processing Flows of Information: From Data Stream to Complex Event Processing [Cugola, Margara]
  - Excellent survey of many different data stream engines!
- Stanford STREAM research project:
  - http://infolab.stanford.edu/stream/
- Aurora and Borealis research projects:
  - http://www.cs.brown.edu/research/aurora/
  - http://www.cs.brown.edu/research/borealis/
- TelegraphCQ research project:
  - http://telegraph.cs.berkeley.edu/