Data Stream Management Systems

Vera Goebel
Department of Informatics, University of Oslo

INF5100, Fall 2017

• What are DSMSs? (terms)
• Why do we need DSMSs? (applications)
• Concepts: Data Model, Query Processing, Windows
• Example: Medical Data Analysis with Esper
Handle Data Streams in DBS?

**Traditional DBS**

SQL Query

<table>
<thead>
<tr>
<th>Query Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Memory</td>
</tr>
<tr>
<td>Disk</td>
</tr>
</tbody>
</table>

**DSMS**

Register CQs

<table>
<thead>
<tr>
<th>Query Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Memory</td>
</tr>
<tr>
<td>Data Stream(s)</td>
</tr>
<tr>
<td>Scratch store (main memory or disk)</td>
</tr>
<tr>
<td>Archive Stored relations</td>
</tr>
</tbody>
</table>

Result (stored)
# Data Management: Comparison - DBS versus DSMS

<table>
<thead>
<tr>
<th>Database Systems (DBS)</th>
<th>DSMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Persistent data (relations) (relatively static, stored)</td>
<td>• Read-only (append-only) data</td>
</tr>
<tr>
<td>• Transactions (ACID properties)</td>
<td>• No transaction management</td>
</tr>
<tr>
<td>• One-time queries</td>
<td>• Transient streams (on-line analysis)</td>
</tr>
<tr>
<td>• Random access</td>
<td>• Continuous queries (CQs)</td>
</tr>
<tr>
<td>• “Unbounded” disk store</td>
<td>• Sequential access</td>
</tr>
<tr>
<td>• Only current state matters</td>
<td>• Bounded main memory</td>
</tr>
<tr>
<td>• No real-time services</td>
<td>• Historical data is important</td>
</tr>
<tr>
<td>• Relatively low update rate</td>
<td>• Real-time requirements</td>
</tr>
<tr>
<td>• Data at any granularity</td>
<td>• Possibly multi-GB arrival rate</td>
</tr>
<tr>
<td>• Assume precise data</td>
<td>• Data at fine granularity</td>
</tr>
<tr>
<td>• Access plan determined by query processor, physical DB design</td>
<td>• Data stale/imprecise</td>
</tr>
<tr>
<td></td>
<td>• Unpredictable/variable data arrival and characteristics</td>
</tr>
</tbody>
</table>

Adapted from [Motawani: PODS tutorial]
DSMS Applications

• Sensor Networks:
  – Monitoring of sensor data from many sources, complex filtering, activation of alarms, aggregation and joins over single or multiple streams

• Network Traffic Analysis:
  – Analyzing Internet traffic in near real-time to compute traffic statistics and detect critical conditions

• Financial Tickers:
  – On-line analysis of stock prices, discover correlations, identify trends

• On-line auctions

• Transaction Log Analysis, e.g., Web, telephone calls, …
Motivation for DSMS

• **Large amounts of interesting data:**
  – deploy transactional data observation points, e.g.,
    • AT&T long-distance: ~300M call tuples/day
    • AT&T IP backbone: ~10B IP flows/day
  – generate automated, highly detailed measurements
    • NOAA: satellite-based measurement of earth geodetics
    • Sensor networks: huge number of measurement points

• **Near real-time queries/analyses**
  – ISPs: controlling the service level
  – NOAA: tornado detection using weather radar data
Motivation for DSMS (cont.)

- Performance of disks:

<table>
<thead>
<tr>
<th></th>
<th>1987</th>
<th>2004</th>
<th>2016 HD (SSD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Performance</td>
<td>1 MIPS</td>
<td>2 Mill. MIPS</td>
<td>1-n Giga/Peta/Exa MIPS</td>
</tr>
<tr>
<td>Memory Size</td>
<td>16 Kbytes</td>
<td>32 Gbytes</td>
<td>1-n TBytes</td>
</tr>
<tr>
<td>Memory Performance</td>
<td>100 usec</td>
<td>2 nsec</td>
<td>2 nsec</td>
</tr>
<tr>
<td>Disc Drive Capacity</td>
<td>20 Mbytes</td>
<td>300 Gbytes</td>
<td>10-16 TBytes</td>
</tr>
<tr>
<td>Disc Drive Performance</td>
<td>60 msec</td>
<td>5.3 msec</td>
<td>2-5 msec HD (SSD: 25-250 usec read, 2 msec write)</td>
</tr>
</tbody>
</table>
Motivation for DSMS (cont.)

• Take-away points:
  – Large amounts of raw data
  – Analysis needed as fast as possible
  – Data feed problem
Application Requirements

- **Data model and query semantics**: order- and time-based operations
  - Selection
  - Nested aggregation
  - Multiplexing and demultiplexing
  - Frequent item queries
  - Joins
  - Windowed queries

- **Query processing**:
  - Streaming query plans must use non-blocking operators
  - Only single-pass algorithms over data streams

- **Data reduction**: approximate summary structures
  - Synopses, digests => no exact answers

- **Real-time reactions** for monitoring applications => active mechanisms

- **Long-running queries**: variable system conditions

- **Scalability**: shared execution of many continuous queries, monitoring multiple streams

- **Stream Mining**
Generic DSMS Architecture

[Golab & Özsu 2003]
DSMS: 3-Level Architecture

**DBS**
- Data feeds to database can also be treated as data streams
- Resource (memory, disk, per-tuple computation) rich
- Useful to audit query results of DSMS
- Supports sophisticated query processing, analyses

**DSMS**
- DSMS at multiple observation points, (voluminous) streams-in, (data reduced) streams-out
- Resource (memory, per tuple computation) limited, esp. at low-level
- Reasonably complex, near real-time, query processing
- Identify what data to populate in DB
Data Models

- **Real-time data stream**: sequence of data items that arrive in some order and may be seen only once.
- **Stream items**: like relational tuples
  - relation-based models, e.g., STREAM, TelegraphCQ; or instantiations of objects
  - object-based models, e.g., COUGAR, Tribeca
- **Window models**:
  - Direction of movement of the endpoints: fixed window, sliding window, landmark window
  - Physical / time-based windows versus logical / count-based windows
  - Update interval: eager (update for each new arriving tuple) versus lazy (batch processing -> jumping window), non-overlapping tumbling windows
Timestamps

• Explicit
  – Injected by data source
  – Models real-world event represented by tuple
  – Tuples may be out-of-order, but if near-ordered can reorder with small buffers

• Implicit
  – Introduced as special field by DSMS
  – Arrival time in system
  – Enables order-based querying and sliding windows

• Issues
  – Distributed streams?
  – Composite tuples created by DSMS?
Time

• Easiest: global system clock
  – Stream elements and relation updates timestamped on entry to system

• Application-defined time
  – Streams and relation updates contain application timestamps, may be out of order
  – Application generates “heartbeat”
    • Or deduce heartbeat from parameters: stream skew, scrambling, latency, and clock progress
  – Query results in application time
Queries - I

- DBS: one-time (transient) queries
- DSMS: continuous (persistent) queries
  - Support persistent and transient queries
  - Predefined and ad hoc queries (CQs)
  - Examples (persistent CQs):
    - Tapestry: content-based email, news filtering
    - OpenCQ, NiagaraCQ: monitor web sites
    - Chronicle: incremental view maintenance
- Unbounded memory requirements
- Blocking operators: window techniques
- Queries referencing past data
Queries - II

• DBS: (mostly) exact query answer
• DSMS: (mostly) approximate query answer
  – Approximate query answers have been studied:
    • Synopsis construction: histograms, sampling, sketches
    • Approximating query answers: using synopsis structures
    • Approximate joins: using windows to limit scope
    • Approximate aggregates: using synopsis structures

• Batch processing
• Data reduction: sampling, synopses, sketches, wavelets, histograms, …
One-pass Query Evaluation

• DBS:
  – Arbitrary data access
  – One/few pass algorithms have been studied:
    • Limited memory selection/sorting: \( n \)-pass quantiles
    • Tertiary memory databases: reordering execution
    • Complex aggregates: bounding number of passes

• DSMS:
  – Per-element processing: single pass to reduce drops
  – Block processing: multiple passes to optimize I/O cost
Query Plan

- DBS: fixed query plans optimized at beginning
- DSMS: adaptive query operators
  - Adaptive plans: Adaptive query plans have been studied:
    - Query scrambling: wide-area data access
    - Eddies: volatile, unpredictable environments
Query Languages & Processing

- Stream query language issues (compositionality, windows)
- SQL-like proposals suitably extended for a stream environment:
  - Composable SQL operators
  - Queries reference relations or streams
  - Queries produce relations or streams
- Query operators (selection/projection, join, aggregation)
- Examples:
  - GSQL (Gigascope)
  - CQL (STREAM)
- Optimization objectives
- Multi-query execution
Query Languages

3 querying paradigms for streaming data:

1. **Relation-based**: SQL-like syntax and enhanced support for windows and ordering, e.g., Esper, CQL (STREAM), StreaQuel (TelegraphCQ), AQuery, GigaScope

2. **Object-based**: object-oriented stream modeling, classify stream elements according to type hierarchy, e.g., Tribeca, or model the sources as ADTs, e.g., COUGAR

3. **Procedural**: users specify the data flow, e.g., Aurora, users construct query plans via a graphical interface

(1) and (2) are declarative query languages, currently, the relation-based paradigm is mostly used.
Windows

• Mechanism for extracting a finite relation from an infinite stream
• Various window proposals for restricting operator scope
  – Windows based on ordering attributes (e.g., time)
  – Windows based on tuple counts
  – Windows based on explicit markers (e.g., punctuations)
  – Variants (e.g., partitioning tuples in a window)
Ordering Attribute Based Windows

• Assumes the existence of an attribute that defines the order of stream elements/tuples (e.g., time)
• Let T be the window length (size) expressed in units of the ordering attribute (e.g., T may be a time window)
• Various possibilities exist:
Tuple Count Based Windows

- Window of size N tuples (sliding, shifting) over the stream
- Problematic with non-unique time stamps associated with tuples
- Ties broken arbitrarily may lead to non-deterministic output
Punctuation Based Windows

• Application inserted “end-of-processing” markers
  – Each data item identifies “beginning-of-processing”
• Enables data item-dependent variable length windows
  – e.g., a stream of auctions
• Similar utility in query processing
  – Limit the scope of query operators relative to the stream
Sample Stream

Traffic (sourceIP -- source IP address
sourcePort -- port number on source
destIP -- destination IP address
destPort -- port number on destination
length -- length in bytes
time -- time stamp
);
Selections, Projections

- Selections, (duplicate preserving) projections are straightforward
  - Local, per-element operators
  - Duplicate eliminating projection is like grouping
- Projection needs to include ordering attribute
  - No restriction for position ordered streams

```sql
SELECT sourceIP, time
FROM Traffic
WHERE length > 512
```
Join Operators

• General case of join operators problematic on streams
  – May need to join arbitrarily far apart stream tuples
  – Equijoin on stream ordering attributes is tractable

• Majority of work focuses on joins between streams with windows specified on each stream

```sql
SELECT A.sourceIP, B.sourceIP
FROM Traffic1 A [window T1], Traffic2 B [window T2]
WHERE A.destIP = B.destIP
```
Aggregation

- **General form:**
  - `select G, F1 from S where P group by G`  
  - Having `F2 op ϑ`
  - `G`: grouping attributes, `F1,F2`: aggregate expressions

- **Aggregate expressions:**
  - **distributive:** sum, count, min, max
  - **algebraic:** avg
  - **holistic:** count-distinct, median
Aggregation & Approximation

• When aggregates cannot be computed exactly in limited storage, approximation may be possible and acceptable

• Examples:
  – select G, median(A) from S group by G
  – select G, count(distinct A) from S group by G
  – select G, count(*) from S group by G having count(*) > f|S|

• Data reduction: use summary structures
  – samples, histograms, sketches …

• Focus of different tutorial
Sampling

- A small random sample S of the data often well-represents all the data
  - Example: select \textit{agg} from R where R.e is odd (n=12)
    
    \begin{itemize}
      \item Data stream: 9 3 5 2 7 1 6 5 8 4 9 1
      \item Sample S: 9 5 1 8
    \end{itemize}

    - If \textit{agg} is avg, return average of odd elements in S
      \textbf{answer: 5}

    - If \textit{agg} is count, return average over all elements \( e \) in S of
      \begin{itemize}
        \item \( n \) if \( e \) is odd
        \item 0 if \( e \) is even
      \end{itemize}
      \textbf{answer: 12*3/4 =9}  \textit{Unbiased!}
Histograms

• Histograms approximate the frequency distribution of element values in a stream

• A histogram (typically) consists of
  – A partitioning of element domain values into buckets
  – A count $C_B$ per bucket $B$ (of the number of elements in $B$)

• Long history of use for selectivity estimation within a query optimizer
Wavelets

- For hierarchical decomposition of functions/signals
- Haar wavelets
  - Simplest wavelet basis => Recursive pairwise averaging and differencing at different resolutions

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Averages</th>
<th>Detail Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[2, 2, 0, 2, 3, 5, 4, 4]</td>
<td>----</td>
</tr>
<tr>
<td>2</td>
<td>[2, 1, 4, 4]</td>
<td>[0, -1, -1, 0]</td>
</tr>
<tr>
<td>1</td>
<td>[1.5, 4]</td>
<td>[0.5, 0]</td>
</tr>
<tr>
<td>0</td>
<td>[2.75]</td>
<td>[-1.25]</td>
</tr>
</tbody>
</table>

Haar wavelet decomposition: [2.75, -1.25, 0.5, 0, 0, -1, -1, 0]
Query Optimization

- DBS: table based cardinalities used in query optimization
  => Problematic in a streaming environment
- Cost metrics and statistics: accuracy and reporting delay vs.
  memory usage, output rate, power usage
- Query optimization: query rewriting to minimize cost metric, adaptive
  query plans, due to changing processing time of operators,
  selectivity of predicates, and stream arrival rates
- Query optimization techniques
  - stream rate based
  - resource based
  - QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability
Disorder in Data Streams

• Many queries over data streams rely on some kind of order on the input data items
  – Can often use more efficient operator implementations if the input is sorted on “interesting attributes” (e.g. aggregates)

• What causes disorder in streams?
  – Items from the same source may take different routes
  – Many sources with varying delays
  – May have been sorted on different attribute

• Sorting a stream may be undesirable

• May be more than one possible interesting order over a stream
  – For example, data items may have creation time and arrival time
  – Sorted on arrival time, but creation time also interesting
Punctuations

• Punctuations embedded in stream denote end of subset of data
  – Unblock blocking operators
  – Reduces state required by stateful operators
• New operator: Punctuate
  – Has special knowledge regarding the input stream
    • timer-based, k-constraints, communication with stream source
  – Emits punctuations in source schema based on special knowledge
• Punctuations can help in two ways:
  • Maintain order – Punctuations unblock sort
    – Similar to approach in Gigascope
    – Order-preserving operators include sort behavior for punctuations
  • Allow disorder – Punctuations define the end of subsets
    – Operators use punctuations, not order, to output results
    – Reduces tuple latency
Query Processing - I

- Continuous query plans:
  - push-based approaches - data is pushed to the DSMS by the source(s)
  - trad.DBS approaches are pull-based, queue problems (overflows)
  - open problems: redesign disk-based data structures and indices

- Processing multiple continuous queries:
  - sharing query plans
  - indexing query predicates

- Distributed query processing:
  - multiple data streams arriving from remote sources
    => distributed optimization strategies
Query Processing - II

(1) Non-blocking operators - 3 techniques for unblocking stream operators:
   • windowing
   • incremental evaluation
   • exploiting stream constraints (punctuations)

(2) Approximate algorithms – if (1) does not work, compact stream summaries may be stored and approximate queries may be run over the summaries
   -> Trade-off: accuracy vs. amount of memory
   Methods of generating synopses: counting methods, hashing methods, sampling methods, sketches, wavelet transformations

(3) Sliding window algorithms:
   • windowed sampling
   • symmetric hash join

(4) On-line data stream mining (single pass): computing stream signatures, decision trees, forecasting, $k$-medians clustering, nearest neighbour queries, regression analysis, similarity detection, pattern matching
Approximate Query Answering Methods

- Sliding windows
  - Only over sliding windows of *recent stream data*
  - Approximation but often more desirable in applications

- Batched processing, sampling and synopses
  - **Batched** if update is fast but computing is slow
    - Compute periodically, not very timely
  - **Sampling** if update is slow but computing is fast
    - Compute using sample data, but not good for joins, etc.
  - **Synopsis** data structures
    - Maintain a small *synopsis* or *sketch* of data
    - Good for querying historical data

- Blocking operators, e.g., sorting, avg, min, etc.
  - **Blocking** if unable to produce the first output until seeing the entire input

[Han 2004]
Query Optimization

- DBS: table based cardinalities used in query optimization => Problematic in a streaming environment
- Cost metrics and statistics: accuracy and reporting delay vs. memory usage, output rate, power usage
- Query optimization: query rewriting to minimize cost metric, adaptive query plans, due to changing processing time of operators, selectivity of predicates, and stream arrival rates
- Query optimization techniques
  - stream rate based
  - resource based
  - QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability
DBS Query Optimization

Statistics Manager:
Periodically collects statistics, e.g., table sizes, histograms

Estimated statistics

Optimizer:
Finds “best” query plan to process this query

Executor:
Runs chosen plan to completion

Which statistics are required

Query

Chosen query plan

[Babu 2004]
Optimizing Continuous Queries

- Continuous queries are long-running
- Stream characteristics can change over time
  - Data properties: Selectivities, correlations
  - Arrival properties: Bursts, delays
- System conditions can change over time

  ➔ Performance of a fixed plan can change significantly over time

  ➔ Adaptive processing: find best plan for current conditions

[Babu 2004]
DSMS Query Optimization

Optimizer: Finds "best" query plan to process this query

Which statistics are required

Profiler: Monitors current stream and system characteristics

Estimated statistics

Reoptimizer: Ensures that plan is efficient for current characteristics

Decisions to adapt

Combined in part for efficiency

Executor: Executes current plan

Query
Optimization Objectives

• Rate-based optimization:
  - Take into account the rates of the streams in the query evaluation tree during optimization
  - Rates can be known and/or estimated

• Maximize tuple output rate for a query
  - Instead of seeking the least cost plan, seek the plan with the highest tuple output rate
Load Shedding

• When input stream rate exceeds system capacity a stream manager can shed load (tuples)

• Load shedding affects queries and their answers

• Introducing load shedding in a data stream manager is a challenging problem

• Random and semantic load shedding
Multi-query Processing

• In traditional multi-query optimization:
  – sharing (of expressions, results, etc.) among queries can lead
  – to improved performance

• Similar issues arise when processing queries on streams:
  – sharing between select/project expressions
  – sharing between sliding window join expressions