Models and Issues in Data Stream Systems
(with changes by CZ)

Rajeev Motwani

Stanford University
(with Brian Babcock, Shivnath Babu, Mayur Datar, and Jennifer Widom)

Data Streams

- **Traditional DBMS** – data stored in finite, persistent data sets
- **New Applications** – data input as continuous, ordered data streams
  - Network monitoring and traffic engineering
  - Telecom call records
  - Network security
  - Financial applications
  - Sensor networks
  - Manufacturing processes
  - Web logs and clickstreams
  - Massive data sets
Sample Applications

- **Network security**
  (e.g., iPolic, NetForensics/Cisco, Niksun)
  - Network packet streams, user session information
  - **Queries:** URL filtering, detecting intrusions & DOS attacks & viruses

- **Financial applications**
  (e.g., Traderbot)
  - Streams of trading data, stock tickers, news feeds
  - **Queries:** arbitrage opportunities, analytics, patterns
  - SEC requirement on closing trades
Executive Summary

- **Data Stream Management Systems (DSMS)**
  - Highlight issues and motivate research
  - Not a tutorial or comprehensive survey

- **Caveats**
  - Personal view of emerging field
  - Stanford STREAM Project bias
  - Cannot cover all projects in detail

---

**DBMS versus DSMS**

- Persistent relations
- One-time queries
- Random access
- “Unbounded” disk store
- Only current state matters
- No real-time services
- Assume precise data

- Transient streams
- Continuous queries
- Sequential access
- Bounded main memory
- History/arrival-order is critical
- Real-time requirements
- Data stale/imprecise
Making Things Concrete

ALICE

BOB

Central Office

DSMS

Central Office

Outgoing (call_ID, caller, time, event)

Incoming (call_ID, callee, time, event)

event = start or end

Query 1 (self-join)

• Find all outgoing calls longer than 2 minutes

SELECT O1.call_ID, O1.caller
FROM Outgoing O1, Outgoing O2
WHERE (O2.time – O1.time > 2
AND O1.call_ID = O2.call_ID
AND O1.event = start
AND O2.event = end)

• Result requires unbounded storage
• Can provide result as data stream
• Can output after 2 min, without seeing end
Query 2 (join)

- **Pair up callers and callees**
  
  ```sql
  SELECT O.caller, I.callee 
  FROM Outgoing O, Incoming I 
  WHERE O.call_ID = I.call_ID 
  ```

- Can still provide **result as data stream**
- Requires **unbounded temporary storage** …
- … unless streams are **near-synchronized**

Query 3 (group-by aggregation)

- **Total connection time** for each caller
  
  ```sql
  SELECT O1.caller, sum(O2.time - O1.time) 
  FROM Outgoing O1, Outgoing O2 
  WHERE (O1.call_ID = O2.call_ID 
  AND O1.event = start
  AND O2.event = end) 
  GROUP BY O1.caller 
  ```

- Join: a very inefficient solution (CZ)
- sum: some window must be specified
Outline of Remaining Talk

- Stream Models and DSMS Architectures
- Query Processing
- Runtime and Systems Issues
- Algorithms
- Conclusion

Data Model

- Append-only
  - Call records
- Updates
  - Stock tickers
- Deletes
  - Transactional data
- Meta-Data
  - Control signals, punctuations

System Internals – probably need all above
Related Database Technology

- **DSMS must use ideas, but none is substitute**
  - Triggers, Materialized Views in Conventional DBMS
  - Main-Memory Databases
  - Distributed Databases
  - Pub/Sub Systems
  - Active Databases
  - Sequence/Temporal/Timeseries Databases
  - Realtime Databases
  - Adaptive, Online, Partial Results

- **Novelty in DSMS**
  - Semantics: input ordering, streaming output, …
  - State: cannot store unending streams, yet need history
  - Performance: rate, variability, imprecision, …

Stream Projects

- Amazon/Cougar (Cornell) – sensors
- **Aurora** (Brown/MIT) – sensor monitoring, dataflow
- Hancock (AT&T) – telecom streams
- Niagara (OGI/Wisconsin) – Internet XML databases
- OpenCQ (Georgia) – triggers, incr. view maintenance
- Stream (Stanford) – general-purpose DSMS
- Tapestry (Xerox) – pub/sub content-based filtering
- **Telegraph** (Berkeley) – adaptive engine for sensors
- Tribeca (Bellcore) – network monitoring
- **ATLAS** (UCLA) – Query power: DB/DS integration.
Aurora/STREAM Overview

- Users issue continuous and ad-hoc queries
- Administrator monitors query execution and adjusts run-time parameters
- Applications register continuous queries

Adaptivity (Telegraph)

- Runtime Adaptivity
- Multi-query Optimization
- Framework – implements arbitrary schemes
Query-Split Scheme (Niagara)

- Aggregate subscription for efficiency
- Split – evaluate trigger only when file updated
- Triggers – multi-query optimization

Outline of Remaining Talk

- Stream Models and DSMS Architectures
- Query Processing
  - Runtime and Systems Issues
  - Algorithms
- Conclusion
Blocking Operators

- **Blocking**
  - No output until entire input seen
  - Streams – input never ends
- **Aggregates** – output “update” stream
- **Set Output** (sort, group-by)
  - Intermediate nodes – try non-blocking analogs
  - Example – j u g g l e for sort [Raman,R,Hellerstein]
  - Punctuations and constraints
- **Join**
  - Sliding-window restrictions

Punctuations [Tucker, Maier, Sheard, Fegaras]

- Assertion about future stream contents
- Unblocks operators, reduces state

State/Index

- Future Work
  - Inserted at source or internal (operator signaling)?
  - Does $P$ unblock $Q$? Exists $P$? Rewrite $Q$?
  - Relation between $P$ and memory for $Q$?

PODS 2002
Impact of Limited Memory

- Continuous streams grow unboundedly
- Queries may require unbounded memory
  - [ABBMW 02]
    - a priori memory bounds for query
    - Conjunctive queries with arithmetic comparisons
    - Queries with join need domain restrictions
    - Impact of duplication elimination
- Open – general queries

Approximate Query Evaluation

- Why?
  - Handling load – streams coming too fast
  - Avoid unbounded storage and computation
  - Ad hoc queries need approximate history
- How? Sliding windows, synopsis, samples, load-shed
- Major Issues?
  - Metric for set-valued queries
  - Composition of approximate operators
  - How is it understood/controlled by user?
  - Integrate into query language
  - Query planning and interaction with resource allocation
  - Accuracy-efficiency-storage tradeoff and global metric
Sliding Window Approximation

• Why?
  – Approximation technique for bounded memory
  – Natural in applications (emphasizes recent data)
  – Well-specified and deterministic semantics

• Issues
  – Extend relational algebra, SQL, query optimization
  – Algorithmic work
  – Timestamps?

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?
Timestamps in JOIN Output

**Approach 1**
- User-specified, with defaults
- Compute output timestamp
- Must output in order of timestamps
- Better for **Explicit** Timestamp
- Need more buffering
- Get precise semantics and user-understanding

**Approach 2**
- Best-effort, no guarantee
- Output timestamp is exit-time
- Tuples arriving earlier more likely to exit earlier
- Better for **Implicit** Timestamp
- Maximum flexibility to system
- Difficult to impose precise semantics

Approximate via Load-Shedding

Handles scan and processing rate mismatch

**Input Load-Shedding**
- Sample incoming tuples
- Use when scan rate is bottleneck
- **Positive** – online aggregation  
[ Hellerstein, Haas, Wang]
- **Negative** – join sampling  
[ Chaudhuri, Motwani, Narasaya]

**Output Load-Shedding**
- Buffer input infrequent output
- Use when query processing is bottleneck
- **Example** – XJoin  
[ Urban, Franklin]
- Exploit synopses
Stream Query Language?

- SQL extension
- Sliding windows as first-class construct
  - Awkward in SQL, needs reference to timestamps
  - SQL-99 allows aggregations over sliding windows
- Sampling/approximation/load-shedding/QoS support?
- Stream relational algebra and rewrite rules
  - Aurora and STREAM
  - Sequence/Temporal Databases

Outline of Remaining Talk

- Stream Models and DSMS Architectures
- Query Processing
- Runtime and Systems Issues
- Algorithms
- Conclusion
DSMS Internals

- **Query plans**: operators, synopses, queues
- **Memory management**
  - Dynamic Allocation – queries, operators, queues, synopses
  - Graceful adaptation to reallocation
  - Impact on throughput and precision
- **Operator scheduling**
  - Variable-rate streams, varying operator/query requirements
  - Response time and QoS
  - Load-shedding
  - Interaction with queue/memory management

Queue Memory and Scheduling

[Babcock, Babu, Datar, Motwani]

- **Goal**
  - Given – query plan and selectivity estimates
  - Schedule – tuples through operator chains
- **Minimize total queue memory**
  - Best-slope scheduling is near-optimal
  - Danger of starvation for some tuples
- **Minimize tuple response time**
  - Schedule tuple completely through operator chain
  - Danger of exceeding memory bound
- **Open** – graceful combination and adaptivity
Queue Memory and Scheduling
[Babcock, Babu, Datar, Motwani]

Input

selectivity = 0.0
S3
Output

selectivity = 0.6
S2

selectivity = 0.2
S1

Net Selectivity

s1
s2
s3

best slope

starvation point

Time

Rate-Based & QoS Optimization

• [Viglas, Naughton]
  – Optimizer goal is to increase throughput
  – Model for output-rates as function of input-rates
  – Designing optimizers?

• Aurora – QoS approach to load-shedding

QoS

% tuples delivered

QoS

Delay

QoS

Output-value

Static: drop-based
Runtime: delay-based
Semantic: value-based

PODS 2002
Outline of Remaining Talk

- Stream Models and DSMS Architectures
- Query Processing
- Runtime and Systems Issues
- Algorithms
- Conclusion

Synopses

- Queries may access or aggregate past data
- Need bounded-memory history-approximation
- Synopsis?
  - Succinct summary of old stream tuples
  - Like indexes/materialized-views, but base data is unavailable
- Examples
  - Sliding Windows
  - Samples
  - Sketches
  - Histograms
  - Wavelet representation
Model of Computation

Increasing time

Memory: \( \text{poly}(1/e, \log N) \)
Query/Update Time: \( \text{poly}(1/e, \log N) \)

\( N \): # tuples so far, or window size
\( e \): error parameter

Many other results …

- **Histograms**
  - V-Opt Histograms
    [Gilbert, Guha, Indyk, Kotidis, Muthukrishnan, Strauss], [Indyk]
  - End-Biased Histograms (Iceberg Queries)
    [Manku, Motwani], [Fang, Shiva, Garcia-Molina, Motwani, Ullman]
  - Equi-Width Histograms (Quantiles)
    [Manku, Rajagopalan, Lindsay], [Khanna, Greenwald]
  - Wavelets
    Seminal work [Vitter, Wang, Iyer] + many others!

- **Data Mining**
  - Stream Clustering
    [Guha, Mishra, Motwani, O’Callaghan]
    [O’Callaghan, Meyerson, Mishra, Guha, Motwani]
  - Decision Trees
    [Domingos, Hulten], [Domingos, Hulten, Spencer]
Conclusion: Future Work

• Query Processing
  – Stream Algebra and Query Languages
  – Approximations
  – Blocking, Constraints, Punctuations

• Runtime Management
  – Scheduling, Memory Management, Rate Management
  – Query Optimization (Adaptive, Multi-Query, Ad-hoc)
  – Distributed processing

• Synopses and Algorithmic Problems

• Systems
  – UI, statistics, crash recovery and transaction management
  – System development and deployment

Thank You!