Models and Issues in Data Stream Systems

Rajeev Motwani

Stanford University

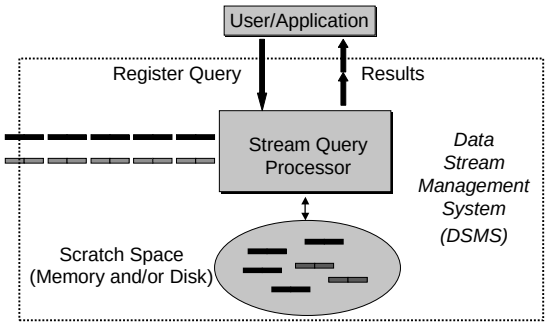
(with Brian Babcock, Shivnath Babu, Mayur Datar, and Jennifer Widom)

STREAM Project Members: Arvind Arasu, Gurmeet Manku, Liadan O'Callaghan, Justin Rosentein, Qi Sun, Rohit Varma

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

Data Stream Management System



Meta-Questions

- Killer-apps
  - Application stream rates exceed DBMS capacity?
  - Can DSMS handle high rates anyway?
- Motivation
  - Need for general-purpose DSMS?
  - Not ad-hoc, application-specific systems?
- Non-Trivial
  - DSMS = merely DBMS with enhanced support for triggers, temporal constructs, data rate mgmt?

Sample Applications

- Network security (e.g., iPolicy, 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

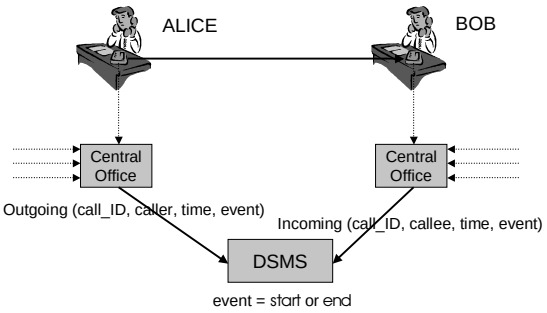
- Data Stream Management Systems (DSMS)
  - Highlight issues and motivate research
  - Not a tutorial or comprehensive survey
- Caveats
  - Personal view of emerging field
  - ☺ Stanford STREAM Project bias
  - ☹ Cannot cover all projects in detail

DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- "Unbounded" disk store
- Only current state matters
- Passive repository
- Relatively low update rate
- No real-time services
- Assume precise data
- Access plan determined by query processor, physical DB design

- Transient streams
- Continuous queries
- Sequential access
- Bounded main memory
- History/arrival-order is critical
- Active stores
- Possibly multi-GB arrival rate
- Real-time requirements
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics

Making Things Concrete



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

```
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

```
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
```

- Cannot provide result in (append-only) stream
  - Output updates?
  - Provide current value on demand?
  - Memory?

Outline of Remaining Talk

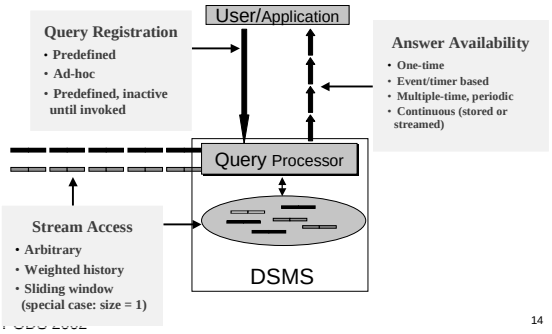
- Stream Models and DSMS Architectures
- Query Processing
- Runtime and Systems Issues
- Algorithms
- Conclusion

Data Model

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

System Internals – probably need all above

Query Model



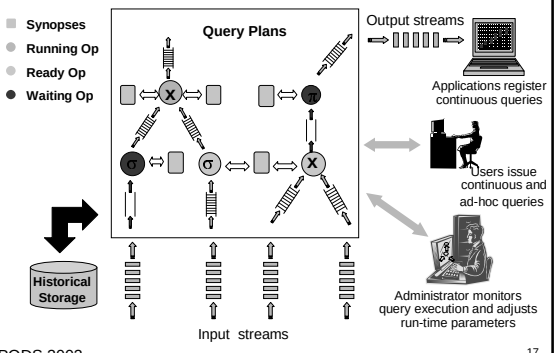
Related Database Technology

- 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, …

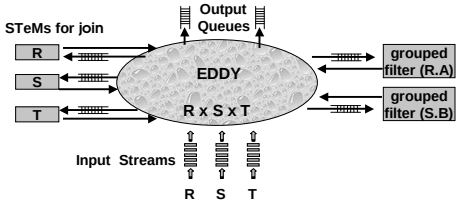
Stream Projects

- 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

Aurora/STREAM Overview

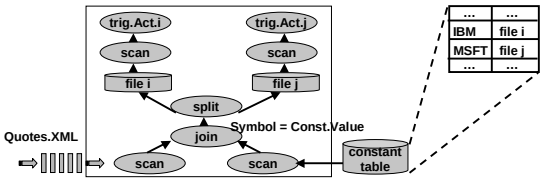


Adaptivity (Telegraph)



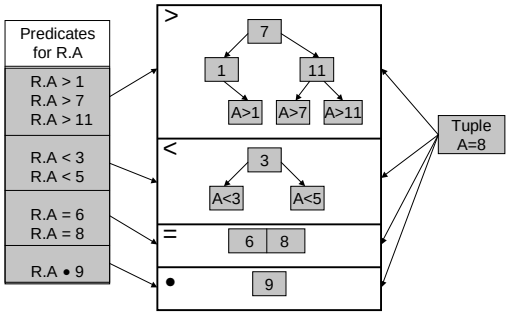
- Runtime Adaptivity
- Multi-query Optimization
- Framework – implements arbitrary schemes

Query-Split Scheme (Niagara)



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

Shared Predicates [Niagara, Telegraph]



Outline of Remaining Talk

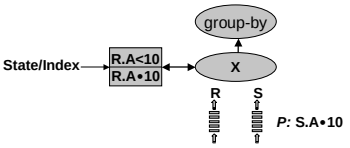
- Stream Models and DSMS Architectures
- Query Processing
- Runtime and Systems Issues
- Algorithms
- Conclusion

Blocking Operators

- Blocking
  - No output until entire input seen
  - Streams – input never ends
- Simple Aggregates – output "update" stream
- Set Output (sort, group-by)
  - Root – could maintain output data structure
  - Intermediate nodes – try non-blocking analogs
  - Example – juggle for sort [Raman,R,Hellerstein]
  - Punctuations and constraints
- Join
  - non-blocking, but intermediate state?
  - sliding-window restrictions

Punctuations [Tucker, Maier, Sheard, Fegaras]

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



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

Constraints

- Schema-level: ordering, referential integrity, many-one joins
- Instance-level: punctuations
- Query-level: windowed join (nearby tuples only)
- [Babu-Widom]
  - Input – multi-stream SPJ query, schema-level constraints
  - Output – plan with low intermediate state for joins
- Future Work
  - Query-level constraints? Combining constraints?
  - Relaxed constraints (near-sorted, near-clustered)
  - Exploiting constraints in intra-operator signaling

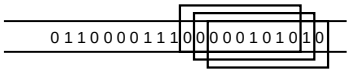
Impact of Limited Memory

- 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

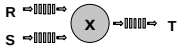


- Why?
  - Approximation technique for bounded memory
  - Natural in applications (emphasizes recent data)
  - Well-specified and deterministic semantics
- Issues
  - Extend relational algebra, SQL, query optimization
  - Algorithmic work
  - Timestamps?

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	Approach 2
• User-specified, with defaults	• Best-effort, no guarantee
• Compute output timestamp	• Output timestamp is exit-time
• Must output in order of timestamps	• Tuples arriving earlier more likely to exit earlier
• Better for Explicit Timestamp	• Better for Implicit Timestamp
• Need more buffering	• Maximum flexibility to system
• Get precise semantics and user-understanding	• Difficult to impose precise semantics

Approximate via Load-Shedding

Handles scan and processing rate mismatch

Input Load-Shedding	Output Load-Shedding
• Sample incoming tuples	• Buffer input infrequent output
• Use when scan rate is bottleneck	• Use when query processing is bottleneck
• Positive – online aggregation [Hellerstein, Haas, Wang]	• Example – XJoin [Urhan, Franklin]
• Negative – join sampling [Chaudhuri, Motwani, Narasaya]	• Exploit synopses

Distributed Query Evaluation

- Logical stream = many physical streams
  - maintain top 100 Yahoo pages
- Correlate streams at distributed servers
  - network monitoring
- Many streams controlled by few servers
  - sensor networks
- Issues
  - Move processing to streams, not streams to processors
  - Approximation-bandwidth tradeoff

Example: Distributed Streams

- Maintain top 100 Yahoo pages
  - Pages served by geographically distributed servers
  - Must aggregate server logs
  - Minimize communication
- Pushing processing to streams
  - Most pages not in top 100
  - Avoid communicating about such pages
  - Send updates about relevant pages only
  - Requires server coordination

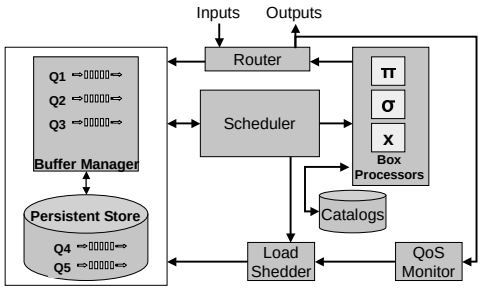
Stream Query Language?

- 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

Aurora Run-time Architecture



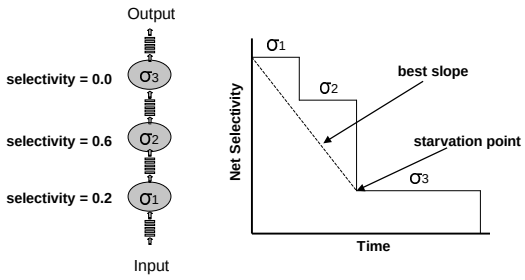
DSMS Internals

- 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]



Precision-Resource Tradeoff

- Resources – memory, computation, I/O
- Global Optimization Problem
  - Input: queries with alternate plans, importance weights
  - Precision: function of resource allocation to queries/operators
  - Goal: select plans, allocate resources, maximize precision
- Memory Allocation Algorithm [Varma, Widom]
  - Model – single query plan, simple precision model
  - Rules for precision of composed operators
  - Non-linear numerical optimization formulation
- Open – Combinatorial algorithm? General case?

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

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

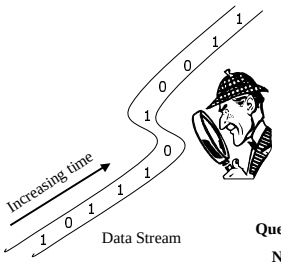
Outline of Remaining Talk

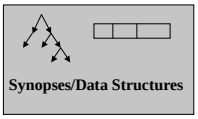
- Stream Models and DSMS Architectures
- Query Processing
- Runtime and Systems Issues
- Algorithms
- Conclusion

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





Memory: poly(1/ε, log N)

Query/Update Time: poly(1/ε, log N)

N: # tuples so far, or window size

ε: error parameter

Sketching Techniques

- [Alon,Matias,Szegedy] frequency moments
- [Feigenbaum etal, Indyk] extended to **Lp** norm
- [Dobra et al] complex aggregates over joins
- Key Subproblem – Self-Join Size Estimation
  - Stream of values from **D = {1,2,…,N}**
  - Let $\mathbf{f_i}$ = frequency of value **i**
  - Self-join size $\mathbf{S = \Sigma f_i^2}$
  - Question – estimating S in small space?

Self-Join Size Estimation

- AMS Technique (randomized sketches)
  - Given $\mathbf{(f_1,f_2,\ldots,f_N)}$
  - $\mathbf{Z_i}$ = random{-1,1}
  - $\mathbf{X = \Sigma f_i Z_i}$ (X incrementally computable)
- Theorem $\mathbf{Exp[X^2] = \Sigma f_i^2}$
  - Cross-terms $\mathbf{f_i Z_i f_j Z_j}$ have 0 expectation
  - Square-terms $\mathbf{f_i Z_i f_i Z_i = f_i^2}$
- Space = $\mathbf{\log (N + \Sigma f_i)}$
- Independent samples $\mathbf{X_k}$ reduce variance

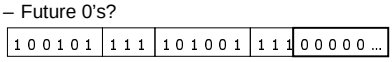
Sliding Window Computations

[Datar, Gionis, Indyk, Motwani]

- Goal: statistics/queries
- Memory: **o(N),** preferably **poly(1/ε, log N)**
- Problem: count/sum/variance, histogram, clustering,…
- Sample Results: **(1+ε)-**approximation
  - Counting: Space **O(1/ε (log N))** bits, Time **O(1)** amortized
  - Sum over [0,R]: Space **O(1/ε log N (log N + log R))** bits, Time **O(log R/log N)** amortized
  - Lp sketches: maintain with **poly(1/ε, log N)** space overhead
  - Matching space lower bounds

Sliding Window Histograms

- Key Subproblem – Counting 1's in bit-stream
- Goal – Space **O(log N)** for window size **N**
- Problem – Accounting for expiring bits
- Idea
  - Partition/track buckets of known count
  - Error in oldest bucket only
  - Future 0's?



Exponential Histograms

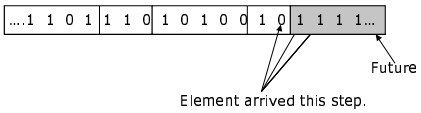
- Buckets of exponentially increasing size
- Between **K/2** and **K/2+1** buckets of each size
- **K = 1/ε** and **ε** = relative error

Exponential Histograms

- Buckets of exponentially increasing size
- Between **K/2** and **K/2+1** buckets of each size
- **K = 1/ε** and **ε** = relative error

K = 2

Bucket sizes = 4,2,2,1,1.



Element arrived this step.

$C_{i-1} + C_{i-2} + \ldots + C_2 + C_1 + 1 >= (K/2)\, C_i$

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!