Building a Data Stream Management System

Prof. Jennifer Widom

Joint project with Prof. Rajeev Motwani and a team of graduate students

http://www-db.stanford.edu/stream

Data Streams

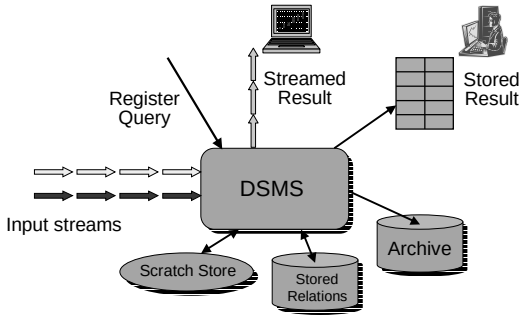
- Continuous, unbounded, rapid, time-varying streams of data elements
- Occur in a variety of modern applications
  - Network monitoring and traffic engineering
  - Sensor networks, RFID tags
  - Telecom call records
  - Financial applications
  - Web logs and click-streams
  - Manufacturing processes
- DSMS = Data Stream Management System

DBMS versus DSMS

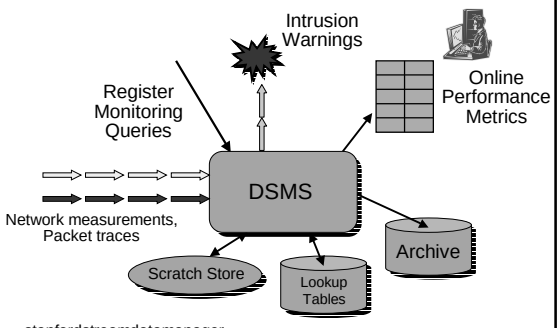
- Persistent relations
- One-time queries
- Random access
- Access plan determined by query processor and physical DB design

- Transient streams (and persistent relations)
- Continuous queries
- Sequential access
- Unpredictable data characteristics and arrival patterns

The (Simplified) Big Picture



(Simplified) Network Monitoring



The STREAM System

- Data streams and stored relations
- Declarative language for registering continuous queries
- Designed to cope with high data rates and query workloads
  - Graceful approximation when needed
  - Careful resource allocation and usage

The STREAM System

- Designed to cope with bursty streams and changing workloads
  - Continuous monitoring of data and system behavior
  - Continuous reoptimization
- Relational, centralized (for now)

Contributions to Date

- Semantics for continuous queries
- Query plans
- Query optimization and dynamic reoptimization
- Exploiting stream constraints
- Operator scheduling
- Approximation techniques
- Resource allocation to maximize precision
- Initial running prototype

Semantics for Continuous Queries

- Abstract: window specification language and relational query language as “black boxes”
- Concrete: SQL-based instantiation for our system; includes syntactic shortcuts, defaults, equivalences
- Goals
  - CQs over multiple streams and relations
  - Exploit relational semantics to the extent possible
  - Easy queries should be easy to write
  - Queries should do what you expect

Concrete Language – CQL

- Relational query language: SQL
- Window spec. language derived from SQL-99
  - Tuple-based, time-based, partitioned
- Syntactic shortcuts and defaults
  - *So easy queries are easy to write and queries do what you expect*
- Equivalences
  - Basis for query-rewrite optimizations
  - Includes all relational equivalences, plus new stream-based ones

CQL Example

Two streams, contrived for example:
Orders (orderID, cost)
Fulfillments (orderID, clerk)

Maximum cost of orders fulfilled by each clerk over his/her last 100 fulfillments:

```
Select F.clerk, Max(O.cost)
From Orders O,
   Fulfillments F [Partition By clerk Rows 100]
Where O.orderID = F.orderID
Group By F.clerk
```


CQL Example

Two streams, contrived for example:
Orders (orderID, cost)
Fulfillments (orderID, clerk)

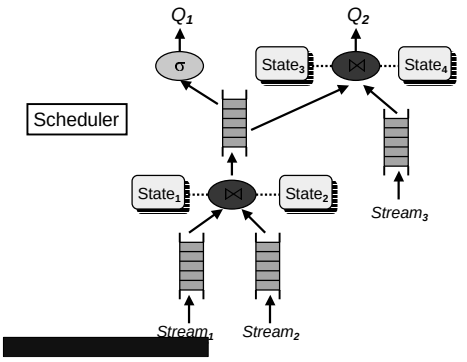
Maximum cost of orders fulfilled by each clerk over his/her last 100 fulfillments (streamed result):

```
Select Istream(F.clerk, Max(O.cost))
From Orders O,
   Fulfillments F [Partition By clerk Rows 100]
Where O.orderID = F.orderID
Group By F.clerk
```

Query Execution

- When a continuous query is registered, generate a query plan
  - Users can also register plans directly
- Plans composed of three main components:
  - Operators (as in most conventional DBMS's)
  - Inter-operator Queues (as in many conventional DBMS's)
  - State (synopses)
- Global scheduler for plan execution

Simple Query Plan



Memory Overhead in Query Processing

- Queues + State
- Continuous queries keep state indefinitely
- Online requirements suggest using memory rather than disk
  - But we realize this assumption is shaky
- Goal: minimize memory use while providing timely, accurate answers

Reducing Memory Overhead

Two main techniques to date

1) Exploit constraints on streams to reduce state
2) Clever operator scheduling to reduce queue sizes

Exploiting Stream Constraints

- For most queries, unbounded memory is required for arbitrary streams
- But streams may exhibit constraints that reduce, bound, or even eliminate state
- Conventional database constraints
  - Redefined for streams
  - "Relaxed" for stream environment

Stream Constraints

- Each constraint type defines adherence parameter *k*
- Clustered(*k*) for attribute *S.A*
- Ordered(*k*) for attribute *S.A*
- Referential-Integrity(*k*) for join $S_1 \rightarrow S_2$

Algorithm for Exploiting Constraints

- Input
  - Any Select-Project-Join query over streams
  - Any set of *k*-constraints
- Output
  - Query execution plan that reduces or eliminates state based on *k*-constraints
  - If constraints violated, get approximate result

Operator Scheduling

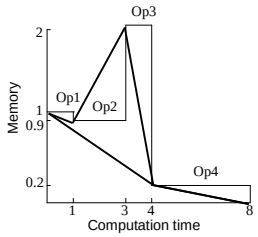
- Global scheduler invokes run method of query plan operators with "timeslice" parameter
- Many possible scheduling objectives: minimize latency, inaccuracy, memory use, computation, starvation, …
  - First scheduler: round-robin
  - ➢ Second scheduler: minimize queue sizes
  - Third scheduler: minimize combination of queue sizes and latency

CHAIN Scheduling Algorithm

- Goal: minimize total queue size for unpredictable, bursty stream arrival patterns
- Proven: within constant factor of any "clairvoyant" strategy for some queries
- Empirical results: large savings over naive strategies for many queries
- But minimizing queue sizes is at odds with minimizing latency

CHAIN Algorithm (contd.)

- Plan progress chart: memory delta per unit time
- Independent scheduling of operators makes mistakes
- Consider chains of operators
- Always schedule ready chain with steepest downslope



Approximation

- Why approximate?
  - Memory requirement too high, even with constraints and clever scheduling
  - Can't process streams fast enough for query load

Approximation (cont'd)

- Static: rewrite queries to add (or shrink) sampling or windows
  - \+ User can participate, predictable behavior
  - – Doesn't consider dynamic conditions
- Dynamic: modify query plan – insert sampling operators, shrink windows, load shedding
  - \+ Adapts to current resource availability
  - – How to convey to user? (major open issue)

The Stream Systems Landscape

- (At least) three general-purpose DSMS prototypes underway
  - STREAM (Stanford)
  - Aurora (Brown, Brandeis, MIT)
  - TelegraphCQ (Berkeley)
- All will be demo'd at SIGMOD '03
- Cooperating to develop stream system benchmark
  - Goal: demonstrate that conventional systems are far inferior for data stream applications

Conclusion: We're building a Data Stream Management System

http://www-db.stanford.edu/stream