Data Stream Management: 30,000 feet

- DBMS world:
  - Static data, dynamic queries
- Data Stream Management world:
  - Dynamic data, static queries

1

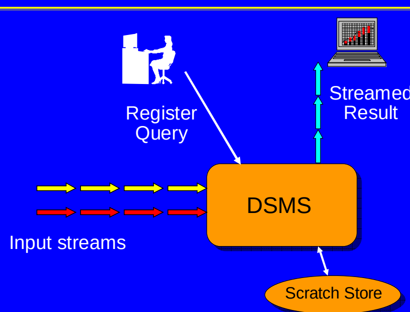
10,000 feet …

- DBMS world:
  - Data is stored, pre-indexed, ~static
  - Queries are ad-hoc and arrive unexpectedly
- Data Stream Management world:
  - Data arrives in continuous, unbounded streams
    - Examples: sensor readings, stock tickers, …
  - Queries are ~static (multiple concurrent "standing queries")
    - Example: alert me when any stock jumps by 5%

"DSMS" = Data Stream Management System

2

5000 feet: DSMS Architecture



3

Research Issues (1/2)

- Languages & formal semantics for data streams and continuous queries
  (what is correct output?)
- Memory requirements & constraints
  (many queries require unbounded memory in worst case)
- Timestamp management & heartbeats
  (data sources tend to have differing latencies)
- Load shedding & approximation
  (keep up with data w/o having to overprovision system)

4

Research Issues (2/2)

- Work sharing
  (concurrent, standing queries --> opportunity to share work)
- Adaptation
  (data characteristics fluctuate, queries persist for long time)
- Operator scheduling
  (data is pushed, not pulled)
- Distributed processing
  (stream sources distributed; improve scalability)

5

Players

- Berkeley [“Telegraph” project]
  – Franklin, Hellerstein
- MIT/Brown/Brandeis [“Aurora” project]
  – Stonebraker, Zdonik, Cherniack
- Stanford [“STREAM” project]
  – Motwani, Widom
- Wisconsin [“Niagara” project]
  – DeWitt, Naughton

6

Players <--> Topics

- STREAM: Languages & formal semantics
- STREAM: Memory requirements & constraints
- STREAM: Timestamp management & heartbeats
- Aurora, STREAM: Load shedding & approximation
- Work sharing: Niagara, Telegraph
- Telegraph, STREAM: Adaptation
- Operator scheduling: Aurora, STREAM
- Aurora: Distributed processing

The Stanford Data Stream Management System

Jennifer Widom

Stanford University

stanfordstreamdatamanager

Formula for a Database Research Project

- Pick a simple but fundamental assumption underlying traditional database systems
  - **Drop it**
- Reconsider all aspects of data management and query processing
  - **Many Ph.D. theses**
  - **Prototype from scratch**

Following the Formula

- We followed this formula once before
  - The **LORE** project
  - Dropped assumption:
    Data has a fixed schema declared in advance
  - Semistructured data (→ XML)
- The **STREAM** Project
  - Dropped assumption:
    First load data, then index it, then run queries
  - Continuous data streams (+ continuous queries)

10

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

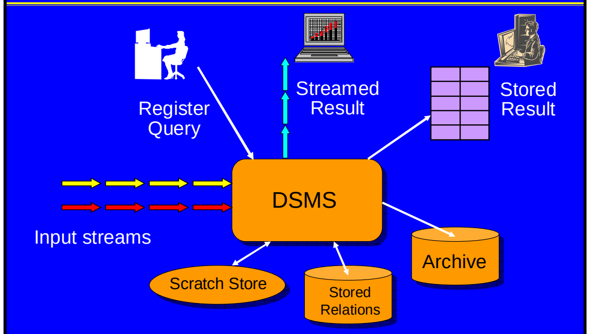
11

DBMS versus DSMS

DBMS	DSMS
Persistent relations	Transient streams (and persistent relations)
One-time queries	Continuous queries
Random access	**Sequential access**
Access plan determined by query processor and physical DB design	**Unpredictable data characteristics and arrival patterns**

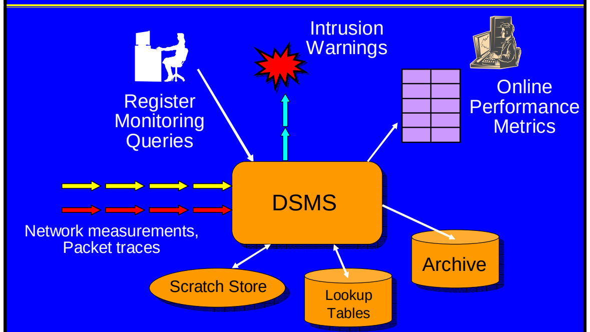
12

The (Simplified) Big Picture



13

(Simplified) Network Monitoring



14

Using Conventional DBMS

- Data streams as relation inserts, continuous queries as triggers or materialized views
- Problems with this approach
  - Inserts are typically batched, high overhead
  - Expressiveness: simple conditions (triggers), no built-in notion of sequence (views)
  - No notion of approximation, resource allocation
  - Current systems don't scale to large # of triggers
  - Views don't provide streamed results

15

The STREAM System

- Data streams and stored relations
- Declarative language for registering continuous queries
- Flexible query plans and execution strategies
- Textual, graphical, and application interfaces
- Relational, centralized (for now)

16

STREAM System Challenges

- Must cope with:
  - Stream rates that may be high,variable, bursty
  - Stream data that may be unpredictable, variable
  - Continuous query loads that may be high, variable

17

STREAM System Challenges

- Must cope with:
  - Stream rates that may be **high**,variable, **bursty**
  - Stream data that may be unpredictable, variable
  - Continuous query loads that may be **high**, variable

Ø **Overload**

18

STREAM System Challenges

- Must cope with:
  - Stream rates that may be high, **variable**, bursty
  - Stream data that may be **unpredictable, variable**
  - Continuous query loads that may be high, **variable**

Ø **Overload**

Ø **Changing conditions**

19

STREAM System Features

- Aggressive sharing of state and computation
- Careful resource allocation and use
- Continuous self-monitoring and reoptimization
- Graceful approximation as necessary

20

Rest of This Talk

- Query language
- Query plans and execution issues
- Coping with overload
- Coping with changing conditions
- **Live system demonstration**

21

Continuous Query Language – CQL

Start with SQL
Then add…

- Streams as new data type
- Continuous instead of one-time semantics
- Windows on streams (derived from SQL-99)
- Sampling on streams (basic)
- Three relation-to-stream operators
  Istream, Dstream Rstream

22

CQL (cont'd)

- Syntactic shortcuts and defaults
  - So easy queries are easy to write
- Equivalences
  - Basis for query-rewrite optimizations
  - Includes all relational equivalences, plus new stream-based ones
- Based on formally-defined abstract semantics

23

CQL Example Query 1

Two streams, contrived for ease of examples:

Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

24

CQL Example Query 1

Two streams, contrived for ease of examples:

Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk "Sue" for customer "Joe"

```
Select Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID And F.clerk = "Sue"
  And O.customer = "Joe"
```

25

CQL Example Query 1

Two streams, contrived for ease of examples:

Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk "Sue" for customer "Joe"

```
Select Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID And F.clerk = "Sue"
  And O.customer = "Joe"
```

26

CQL Example Query 1

Two streams, contrived for ease of examples:

Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk "Sue" for customer "Joe"

```
Select Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID And F.clerk = "Sue"
  And O.customer = "Joe"
```

27

CQL Example Query 1

Two streams, contrived for ease of examples:

Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk "Sue" for customer "Joe"

```
Select  Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID  And F.clerk = "Sue"
  And O.customer = "Joe"
```

28

CQL Example Query 1

Two streams, contrived for ease of examples:

Orders (orderID, customer, cost)
Fulfillments (orderID, clerk)

Total cost of orders fulfilled over the last day by clerk "Sue" for customer "Joe"

```
Select  Sum(O.cost)
From Orders O, Fulfillments F [Range 1 Day]
Where O.orderID = F.orderID  And F.clerk = "Sue"
  And O.customer = "Joe"
```

29

CQL Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
Select F.clerk, Max(O.cost)
From Orders O,
  Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```

30

CQL Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
Select F.clerk, Max(O.cost)
From Orders O,
  Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```

31

CQL Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
Select F.clerk, Max(O.cost)
From Orders O,
  Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```

32

CQL Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
Select F.clerk, Max(O.cost)
From Orders O,
  Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```

33

CQL Example Query 2

Using a 10% sample of the Fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
Select F.clerk, Max(O.cost)
From Orders O,
  Fulfillments F [Partition By clerk Rows 5] 10% Sample
Where O.orderID = F.orderID
Group By F.clerk
```

34

CQL Example: Result Type

Simpler version of Example Query 2:

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

- Result is a relation, updated as stream elements arrive

35

CQL Example: Result Type

Simpler version of Example Query 2:

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

- Streamed result: Emits <clerk,max> stream element whenever max changes for a clerk (or new clerk)

36

CQL Example Query 4

Relation **CurPrice(stock, price)**

Select stock, Avg(price)
From Istream(CurPrice) [Range 1 Day]
Group By stock

- Average price over last day for each stock
- *Istream* provides history of *CurPrice*
- Window on history (back to relation), group and aggregate

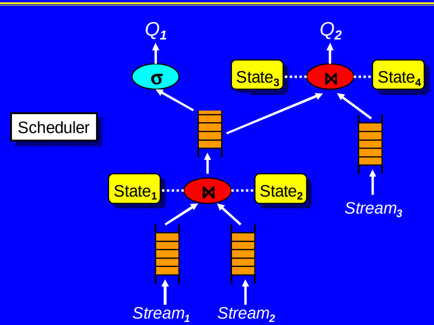
37

Query Execution

- When a continuous query is registered, generate a query plan
  - New plan merged with existing plans
  - Users can also create & manipulate plans directly
- Plans composed of three main components:
  - Operators
  - Queues (input and inter-operator)
  - State (windows, operators requiring history)
- Global scheduler for plan execution

38

Simple Query Plan



39

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

40

Reducing Memory Overhead

1) Exploit constraints on streams to reduce state
2) Enable state sharing within and across queries
3) Specialized operator scheduling to reduce queue sizes

41

Exploiting Stream Constraints

- For many queries, large or unbounded state is required for arbitrary streams

42

Exploiting Stream Constraints

- For many queries, large or unbounded state is required for arbitrary streams
- But streams may exhibit constraints that reduce, bound, or even eliminate state
  - **Clustered**
  - **Ordered**
  - **Stream-based referential integrity**

  Relaxed version: *k*-constraints

43

Stream Constraints: Simple Example

Orders (orderID, customer, cost)
Fulfillments (orderID, portion, clerk)

If **Fulfillments** is *k*-clustered on **orderID**, can infer when to discard **Orders**

44

Exploiting Constraints

- Continuously monitor streams to identify *k*-constraints relevant to queries
- Query execution plans reduce or eliminate state based on *k*-constraints
- If constraints violated, get approximate result

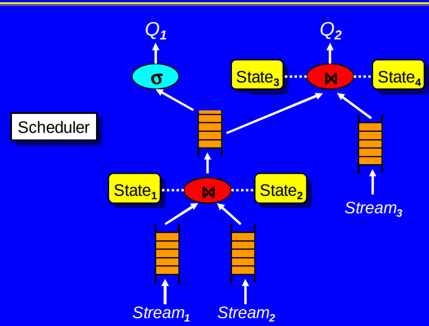
45

State Sharing

- Baseline: Input streams shared by all queries
  - Maintain maximum window
- Subplans and synopses also can be shared
  - Currently must hook up manually
- Sophisticated techniques for sharing and memory minimization in sliding-window aggregates

46

Reminder: Query Plans



47

Operator Scheduling

- Global scheduler invokes **run** method of query plan operators with “timeslice” parameter
- Many possible scheduling objectives: minimize latency, memory use, computation, inaccuracy, starvation, …
  1) Round-robin
  2) Minimize queue sizes
  3) Minimize combination of queue sizes and latency
  4) Parallel versions of above

48

Coping with Overload

- "Load-shedding" ≈ discarding tuples
- Goal: deliver best possible approximate answer while not falling behind
- What is definition of "best"?
  - Maximum subset
  - Maximum random sample
- We have techniques with provable guarantees for specific query types
  - Extremely hard problem for general plans

49

Coping with Changing Conditions

- Continuous queries are long-running; conditions may change
  - Data characteristics, arrival characteristics, query load, available resources, system conditions, …
- Solution: self-monitoring and adaptivity
  - We already saw one example (what was it?)
  - Other results:
    - Adaptive operator reordering
    - Adaptive caching

50

A Note on Time

- All stream elements have timestamps
  - Necessary for time-based windows
  - Necessary for consistent well-defined semantics over multiple streams and updatable relations
- Basic correctness requirement: query processor must see stream elements in timestamp order
- Easy when time is centralized system clock
  - Stream elements timestamped on entry to system

51

Application-Defined Time

- Streams may contain application timestamps
  - Sensor readings, financial transactions, etc.
- Elements may arrive out of order at DSMS
  - Distributed streams with time skew among them
  - Latency reaching DSMS
  - Reordering on transmission channel
- Our solution: heartbeats
  - Provided by application or deduced from measured parameters (skew, latency, etc.)

52

The Stream Systems Landscape

- (At least) three general-purpose DSMS prototypes underway
  - STREAM (Stanford)
  - Aurora - Borealis (Brown, Brandeis, MIT)
  - TelegraphCQ - HiFi (Berkeley)
- Stream system benchmark
  - Main goal: demonstrate that conventional systems are far inferior for data stream applications

53

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

Contributors: **Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Justin Rosenstein, Zubin Wang**