

CIS 700/005 Networking Meets Databases

Boon Thau Loo
Spring 2007
Lecture 17

Several slides are courtesy of Minos Garofalakis, Rajeev Motwani, and Raymond Pan

Announcements

- Project status report due 23rd March
 - See <http://www.cis.upenn.edu/~boonloo/cis700-sp07/project.html#status> for more information
 - Up to 2 pages in length
 - Reusable for final report
 - Email me or see me during OH if require help

Stream Overview

- Active area of research in database community (2001-)
- Relevance to class:
 - Understand practical use in Internet-scale network monitoring, and sensor network data processing
- We will study:
 - Language, data model and applications, STREAM
 - Systems: Aurora, Borealis
 - Optimizations (centralized and distributed)
- Optional reading:
 - TelegraphCQ, HiFI, Gigascope
 - Processing streams and archived data, history-enhanced monitoring

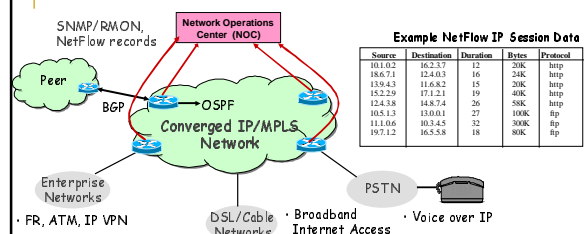
Commercial Impact

- Amalgamated Insight (<http://www.aminsight.com>)
 - TelegraphCQ and HiFI (Berkeley)
- Streambase (<http://www.streambase.com>)
 - Aurora and Borealis (Brandeis/Brown/MIT)
 - Financial services, eBusiness, telecommunications, networking, and government/military setting (command and control)
- Gigascope
 - Network monitoring in AT&T

Data-Stream Management

- Traditional DBMS – data stored in finite, persistent data sets
- Data Streams – distributed, continuous, unbounded, rapid, time varying, noisy, . . .
- Data-Stream Management – variety of modern applications
 - Network monitoring and traffic engineering
 - Telecom call-detail records
 - Network security
 - Financial applications
 - Sensor networks
 - Manufacturing processes
 - Web logs and clickstreams
 - Massive data sets
 - Inventory management

Networks Generate Massive Data Streams

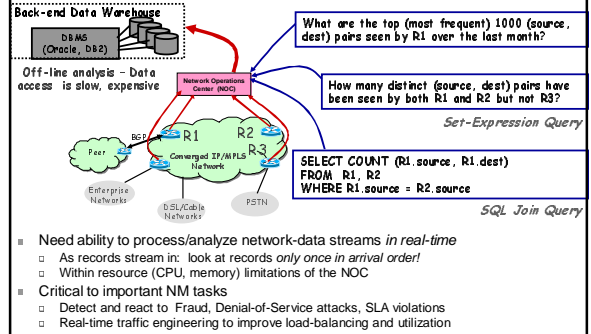


- SNMP/RMON/NetFlow data records arrive 24x7 from different parts of the network
- Truly massive streams arriving at rapid rates
 - AT&T collects 600-800 GigaBytes of NetFlow data each day!
- Typically shipped to a back-end data warehouse (off site) for off-line analysis

Packet-Level Data Streams

- Single 2Gb/sec link; say avg packet size is 50bytes
- Number of packets/sec = 5 million
- Time per packet = 0.2 microsec
- If we only capture header information per packet: src/dest IP, time, no. of bytes, etc. – at least 10bytes.
 - Space per second is 50Mb
 - Space per day is 4.5Tb per link
 - ISPs typically have hundred of links!
- Analyzing packet content streams – whole different ballgame!!

Real-Time Data-Stream Analysis



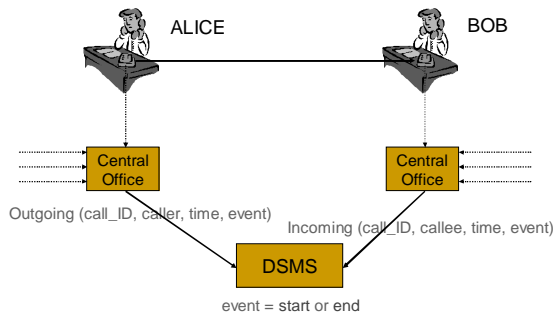
IP Network Data Processing

- Traffic estimation
 - How many bytes were sent between a pair of IP addresses?
 - What fraction network IP addresses are active?
 - List the top 100 IP addresses in terms of traffic
- Traffic analysis
 - What is the average duration of an IP session?
 - What is the median of the number of bytes in each IP session?
- Fraud detection
 - List all sessions that transmitted more than 1000 bytes
 - Identify all sessions whose duration was more than twice the normal
- Security/Denial of Service
 - List all IP addresses that have witnessed a sudden spike in traffic
 - Identify IP addresses involved in more than 1000 sessions

DBMS versus SDSMS

- | | |
|--|--|
| <ul style="list-style-type: none"> 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 | <ul style="list-style-type: none"> 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 |
|--|--|

Queries by Example



Example 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

## Example 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

## Example Query 3 (group-by aggregation)

- Total connection time for each caller
 

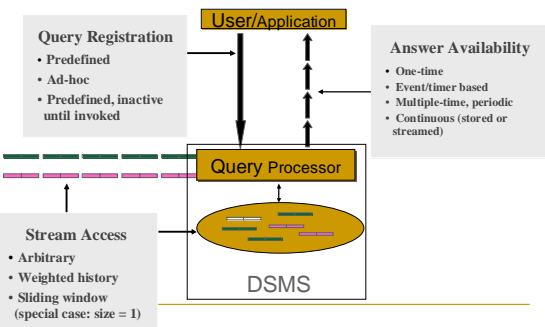
```
SELECT O1.caller, sum(O2.time - O1.time)
FROM Outgoing O1, Outgoing O2
WHERE (O1.call_ID = O2.call_ID
 AND O1.event = start
 AND O2.event = end)
GROUP BY O1.caller
```
- Cannot provide result in (append-only) stream
  - Output updates?
  - Provide current value on demand?
  - Memory?

## Data Model

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

System Internals – probably need all above

## DSMS at a High Level



## 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
  - Real-time 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, ...

## Types of Queries

- One-time queries and Continuous queries
  - One-time queries
    - Evaluated once over a point-in-time snapshot of data set
  - Continuous queries
    - Evaluated continuously as data streams continue to arrive
    - May be stored and updated as new data arrives, or may produce data streams themselves

## Types of Queries

- Predefined and Ad hoc queries
  - Predefined
    - Supplied to data stream management system before any relevant data has arrived
    - Usually *continuous queries*
    - Scheduled one-time queries possible
  - Ad hoc
    - Can be either one-time or continuous queries
    - Complicates design of data stream management system
      - Data not known in advance for purposes of query optimization,
      - Correctly answering query may require referencing previously arrived data streams and potentially been discarded

## Challenges in Stream Query Processing

- Unbounded Memory Requirements
- Approximate Query Answering
- Sliding Windows
- Batch Processing, Sampling, and Synopses
- Blocking Operators
- Queries Referencing Past Data

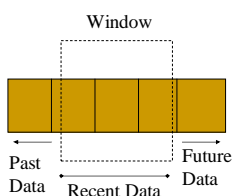
## Unbounded Memory Requirements

- Since data streams are potentially unbounded in size, amount of storage required to compute exact answer to the query may grow without bound
- External memory algorithms for handling data sets larger than main memory cannot be used.
  - Do not support continuous queries
  - Too slow real-time response
- With new data constantly arriving even as old data is being processed, amount of computation time per data element must be low
- Interested in algorithms that are able to confine themselves to main memory without accessing disk

## Approximate Query Answering

- What if we limited ourselves to bounded amount of memory?
  - Exact answers no longer possible
  - Aim for high-quality approximate answers
- Techniques for data reduction and synopsis construction
  - Sketches
  - Random sampling
  - Histograms
  - Wavelets

## Sliding Windows



- Evaluate query over sliding window of recent data from streams
- Attractive Properties
  - Well-defined and understood
  - Deterministic so there is no danger that bad random choices will produce bad approximation
  - Emphasizes recent data, which in many real-world applications is more important than old data

## Sliding Windows

- Research Issues
  - How do we define timestamps over streams to facilitate use of windows?
  - How do we implement sliding window queries?
  - What is their impact on query optimization?
  - If window is too big to fit in main memory, how can we give approximate answers using only available memory?

## Sliding Windows

- Differences in sequence and temporal DB and stream computation model
  - Temporal DB
    - Concerned with full history of each data value over time
    - Stream system concerned with processing new data elements on-the-fly
  - Sequence DB
    - Attempt to produce query plans that allow for stream access.
    - A single scan of input data is sufficient to evaluate plan, and amount of memory required for plan evaluation is constant, independent of data.
    - Assumes that DB system has control over which sequence to process tuples from next (e.g., merging multiple sequences, which cannot be assumed in stream system)

## Batch Processing, Sampling, and Synopses

- Do not process data elements as it arrives
- Resort to sampling or batch processing technique to speed up query execution
- Framework
  - Query answered using data structures that can be maintained incrementally
  - Data structure supports two operations
    - update(tuple) updates data structure as each new data element arrives
    - computeAnswer() produces new or updated results to query
- Best case scenario is that both operations are fast relative to arrival rate of elements in data streams – no special techniques needed

## Batch Processing

- Case 1:
  - update(tuple) is fast but computeAnswer() is slow
- Data elements buffered as they arrive
- Answer to query is computed periodically as time permits
- Does not cause any uncertainty about accuracy of answer, sacrificing timeliness instead.
- Good when data streams are bursty

## Sampling

- Case II:
  - computeAnswer() fast, but update(tuple) slow
- Some tuples skipped altogether so query is evaluated over sample of data stream rather than over entire data stream.
- Give confidence bounds on degree of error introduced by sampling process
- For many situations and queries involving joins, it is not reliable

## Synopsis Data Structures

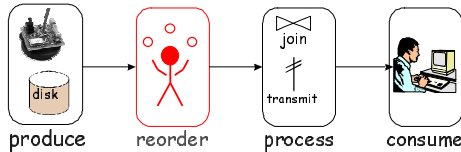
- Case III:
  - computeAnswer() fast, and update(tuple) fast
- Used for queries where no exact data structure with desired properties exists
- Approximate data structure
  - Maintains small synopsis or sketch of data rather than exact representation,
  - Computation per data element is low.

## Blocking Operators

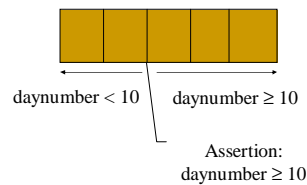
- Query operator that is unable to produce the first tuple of its output until it has seen its entire input.
  - (e.g., sorting, aggregation operators like SUM)
- Since input streams may be infinite,
  - When does a block operator output its results? (if ever?)
- Block operator is root (output) of query plan:
  - Aggregation: incremental generate updates to aggregates
  - Sort: maintain output data-structure containing sorted results

## Blocking Operators as Interior nodes

- Replace interior nodes with non-blocking analogs
  - Juggle operator is a non-blocking version of sort.
  - It aims to locally reorder a data stream so that tuples that come earlier in desired sort order are produced before tuples that come later in sort order, although some tuples may be delivered out of order



## Punctuations for Blocking Operators



- Augmenting data streams with assertions about what can and cannot appear in remainder of data stream
  - Assertions (punctuations) interleaved with data elements in stream
  - Example with assertion "for all future tuples, daynumber  $\geq 10$ "
    - Aggregation operator that was grouping by daynumber could stream out its answers for all daynumbers  $< 10$
    - Join operator could discard all its saved state relating to previously-seen tuples in joining stream with daynumber  $< 10$

## Queries Referencing Past Data

- Ad hoc queries that are issued after some data has already been discarded may be impossible to answer accurately
- Solutiona:
  - Only allow ad hoc queries that reference future data. It may be acceptable in some applications
  - Maintain summaries of data streams (synopses or aggregates) that can approximate answers to future ad hoc queries
- Analogous problem in database indexes and materialized views:
  - Traditional DB: We can still get the right answer at higher cost if no index present.
  - Stream model: if no summary structure present, we cannot get the answer

## Stanford's Proposal for DSMS

- STREAM (STanford StREam DatA Manager)**
- Query Language for a DSMS
- Timestamps in Streams
- STREAM Architecture
- Algorithms for Synopsis Generation

## Query Language for a DSMS

```
SELECT AVG(S.minutes)
FROM Calls S [PARTITION BY
 S.customer_id
 ROWS 10 PRECEDING
 WHERE S.type='Long
 Distance']
```

- Modified version of SQL
- Allowed the FROM clause to refer to streams as well as relations
- Allowing optional window specification to be provided after a stream that is supplied into a query's FROM clause
- Sliding window requires an ordering of data stream elements, using "implicit timestamp" attached to each data element
- Example: Compute average call length, considering only ten most recent long-distance calls placed by each customer

## 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

## User-defined timestamps in Streams

```
SELECT *
FROM S1[ROWS 1000 PRECEDING],
 S2[ROWS 100 PRECEDING]
WHERE S1.A = S2.B
```

**Output tuple will have same timestamp as S1**

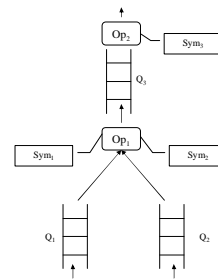
- User specifies as part of query what timestamp is to be assigned to tuples resulting from join of multiple streams
- Order in which streams are listed in FROM clause of query represents a prioritization of streams
- Implementation can be difficult
  - Recent S1 tuple joins with S2 tuple
  - Future S2 tuple join with older S1 tuple
  - How to ensure ordering out output tuples?

## Timestamps in Streams

### ■ “Best-effort”

| Clause            | Meaning                                                                     |
|-------------------|-----------------------------------------------------------------------------|
| ROWS 10 PRECEDING | Window consisting of previous 10 tuples, strictly sorted by timestamp order |
| ROWS 10 RECENT    | DSMS allowed to use its own ordering to produce window                      |

## Stanford STREAM Architecture



- Query execution plans consist of operators connected by queues
- Operators scheduled for execution by central scheduler
  - During execution, operator:
    - reads data from its input queues,
    - updates synopsis structure
    - compute results
    - writes results to output queues
  - Period of execution of operator determined dynamically by scheduler and operator returns control back to scheduler once period expires

## Memory Adaptive Operators

- To handle stream data characteristic fluctuations, operators are adaptive (primarily to memory)
- Trading accuracy for memory
  - Operator maximizes accuracy of output based on size of available memory
  - Handles dynamic changes in size of its available memory
  - Examples:
    - For a sliding window join, the larger the window, the better the approximation
    - Limited-size hash tables for duplicate elimination
    - Sampling

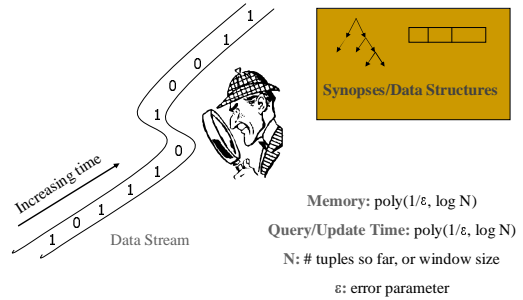
## Scheduling Multiple Query Plans

- Scheduler needs to provide *rate synchronization* within operators and pipelined operators in query plans
- Time-varying arrival rates of data streams and time-varying output rates of operators complicate matters
- Need to take into account
  - Memory allocation across operators
  - Management of buffers for incoming streams
  - Availability of synopses on disk (instead of memory)
  - Performance requirements of individual queries
- Later paper on Aurora: QoS approach to load-shedding on input and output streams

## Algorithms for 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
  - Random sampling
  - Sliding Windows
  - Sketches
  - Histograms
  - Wavelet representation

## Model of Computation



## Sketches

- $S = (x_1, \dots, x_n)$ : sequence of elements where each  $x_i$  belongs to the domain  
 $D = \{1, \dots, d\}$   
 $m_i = |\{j \mid x_j = i\}|$ : number of occurrences of value  $i$  in  $S$
- $F_k = \sum_{i=1}^d m_i^k$ :  $k^{\text{th}}$  frequency moment  
 $F_\infty^* = \max_i m_i$
- Building summary of data stream using small amount of memory
  - Makes it possible to estimate answer to certain queries (like "distance" queries) over data set
  - $F_0$  is number of distinct values in  $S$
  - $F_1$  is the length of  $S$
  - $F_2$  is the self-join size (Gini's index of homogeneity)
  - $F_\infty$  is the most frequent item's multiplicity

## Histograms

- **Captures distribution of values in a dataset**
  - Query size estimation, approximate query answering, and data mining
- **V-Optimal Histogram:**
  - Approximate distribution of a set of values by a piecewise-constant function so as to minimize the sum of squared error.
- **Equi-Width Histograms:**
  - Partition the domain into buckets such that the number of values falling into each bucket is uniform across all buckets. They maintain quantiles for the underlying data distribution as the bucket boundaries.
- **End-Biased Histograms:**
  - Maintain exact counts of items that occur with frequency above a threshold, and approximate other counts by an uniform distribution. [*Iceberg Query*]

## Conclusion

- Survey paper on streams
  - Data model
  - Query model
  - Different research issues
  - STREAM (<http://infolab.stanford.edu/stream/>)
- Issues to think about:
  - How can we extend PIER/P2 to support stream language/execution/optimization techniques?
- Next paper:
  - Aurora stream processing system (VLDBJ '03)