DSMS – Overview

• Introduction:
  – What are DSMS? (terms)
  – DSMS vs. DBMS
  – Why do we need DSMS? (applications)

• Concepts and issues:
  – Architecture(s)
  – Data modeling
  – Query processing and optimization
  – Data Reduction & Stream Mining

• Examples

• Summary: Open issues & conclusions
Handle Data Streams in DBS?

Traditional DBS

- SQL Query
- Main Memory
- Disk
- Query Processing

DSMS

- Register CQs
- Main Memory
- Data Stream(s)
- Scratch store (main memory or disk)
- Archive Stored relations
- Result (stored)
- Query Processing
Data Management

• Traditional DBS:
  – stored sets of relatively static records with no pre-defined notion of time
  – good for applications that require persistent data storage and complex querying

• DSMS:
  – support on-line analysis of rapidly changing data streams
  – data stream: real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items, too large to store entirely, not ending
  – continuous queries
# Data Management: Comparison - DBS versus DSMS

## Database Systems (DBS)
- Persistent relations  
  (relatively static, stored)
- One-time queries
- Random access
- “Unbounded” disk store
- Only current state matters
- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

## DSMS
- Transient streams  
  (on-line analysis)
- Continuous queries (CQs)
- Sequential access
- Bounded main memory
- Historical data is important
- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics
Related DBS Technologies

• Continuous queries
• Active DBS (triggers)
• Real-time DBS
• Adaptive, on-line, partial results
• View management (materialized views)
• Sequence/temporal/timeseries DBS
• Main memory DBS
• Distributed DBS
• Parallel DBS
• Pub/sub systems
• Filtering systems
• …

=> Must be adapted for DSMS!
DSMS Applications

- **Sensor Networks:**
  - Monitoring of sensor data from many sources, complex filtering, activation of alarms, aggregation and joins over single or multiple streams

- **Network Traffic Analysis:**
  - Analyzing Internet traffic in near real-time to compute traffic statistics and detect critical conditions

- **Financial Tickers:**
  - On-line analysis of stock prices, discover correlations, identify trends

- **On-line auctions**

- **Transaction Log Analysis**, e.g., Web, telephone calls, …
Data Streams - Terms

- A **data stream** is a (potentially unbounded) sequence of tuples
- **Transactional data streams**: log interactions between entities
  - Credit card: purchases by consumers from merchants
  - Telecommunications: phone calls by callers to dialed parties
  - Web: accesses by clients of resources at servers
- **Measurement data streams**: monitor evolution of entity states
  - Sensor networks: physical phenomena, road traffic
  - IP network: traffic at router interfaces
  - Earth climate: temperature, moisture at weather stations
Motivation for DSMS

• Large amounts of interesting data:
  – deploy transactional data observation points, e.g.,
    • AT&T long-distance: ~300M call tuples/day
    • AT&T IP backbone: ~10B IP flows/day
  – generate automated, highly detailed measurements
    • NOAA: satellite-based measurement of earth geodetics
    • Sensor networks: huge number of measurement points
Motivation for DSMS (cont.)

• Near real-time queries/analyses
  – ISPs: controlling the service level
  – NOAA: tornado detection using weather radar data

• Traditional data feeds
  – Simple queries (e.g., value lookup) needed in real-time
  – Complex queries (e.g., trend analyses) performed off-line
Motivation for DSMS (cont.)

- Performance of disks:

<table>
<thead>
<tr>
<th></th>
<th>1987</th>
<th>2004</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Performance</td>
<td>1 MIPS</td>
<td>2,000,000 MIPS</td>
<td>2,000,000 x</td>
</tr>
<tr>
<td>Memory Size</td>
<td>16 Kbytes</td>
<td>32 Gbytes</td>
<td>2,000,000 x</td>
</tr>
<tr>
<td>Memory Performance</td>
<td>100 usec</td>
<td>2 nsec</td>
<td>50,000 x</td>
</tr>
<tr>
<td>Disc Drive Capacity</td>
<td>20 Mbytes</td>
<td>300 Gbytes</td>
<td>15,000 x</td>
</tr>
<tr>
<td>Disc Drive Performance</td>
<td>60 msec</td>
<td>5.3 msec</td>
<td>11 x</td>
</tr>
</tbody>
</table>

Source: Seagate Technology Paper: "Economies of Capacity and Speed: Choosing the most cost-effective disc drive size and RPM to meet IT requirements"
Motivation for DSMS (cont.)

• The PingER project:
  – Believed to be the most extensive Internet end-to-end performance monitoring tool in the world

Motivation for DSMS (cont.)

Disk Throughput

Motivation for DSMS (cont.)

• Take-away points:
  – Large amounts of raw data
  – Analysis needed as fast as possible
  – Data feed problem
Application Requirements

• **Data model and query semantics:** order- and time-based operations
  – Selection
  – Nested aggregation
  – Multiplexing and demultiplexing
  – Frequent item queries
  – Joins
  – Windowed queries

• **Query processing:**
  – Streaming query plans must use non-blocking operators
  – Only single-pass algorithms over data streams

• **Data reduction:** approximate summary structures
  – Synopses, digests => no exact answers

• **Real-time reactions** for monitoring applications => active mechanisms

• **Long-running queries:** variable system conditions

• **Scalability:** shared execution of many continuous queries, monitoring multiple streams

• **Stream Mining**
Generic DSMS Architecture

[Diagram showing the architecture with components such as Input Monitor, Working Storage, Summary Storage, Static Storage, Query Processor, and Output Buffer, with arrows indicating streaming inputs and outputs, and updates to static data and user queries.]
DSMS: 3-Level Architecture

**DBS**
- Data feeds to database can also be treated as data streams
- Resource (memory, disk, per-tuple computation) rich
- Useful to audit query results of DSMS
- Supports sophisticated query processing, analyses

**DSMS**
- DSMS at multiple observation points, (voluminous) streams-in, (data reduced) streams-out
- Resource (memory, per tuple computation) limited, esp. at low-level
- Reasonably complex, near real-time, query processing
- Identify what data to populate in DB

---

*VLDB 2003 Tutorial [Koudas & Srivastava 2003]*
Data Models

- **Real-time data stream**: sequence of data items that arrive in some order and may be seen only once.
- **Stream items**: like relational tuples
  - relation-based models, e.g., STREAM, TelegraphCQ; or instanciations of objects
  - object-based models, e.g., COUGAR, Tribeca
- **Window models:**
  - Direction of movement of the endpoints: fixed window, sliding window, landmark window
  - Physical / time-based windows versus logical / count-based windows
  - Update interval: eager (update for each new arriving tuple) versus lazy (batch processing -> jumping window), non-overlapping tumbling windows
Relation: Tuple Set or Sequence?

• Traditional relation = set/bag of tuples
• Tuple sequences:
  – Temporal databases: multiple time orderings
  – Sequence databases: integer “position” -> tuple
• DSMS:
  – Ordering domains: Gigascope, Hancock
  – Position ordering: Aurora, STREAM
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?
Time

• Easiest: global system clock
  – Stream elements and relation updates timestamped on entry to system

• Application-defined time
  – Streams and relation updates contain application timestamps, may be out of order
  – Application generates “heartbeat”
    • Or deduce heartbeat from parameters: stream skew, scrambling, latency, and clock progress
  – Query results in application time
Update: Modifications or Appends?

• Traditional relational updates: arbitrary data modifications

• Append-only relations have been studied:
  – Tapestry: emails and news articles
  – Chronicle data model: transactional data

• DSMS:
  – Streams-in, stream-out: Aurora, Gigascope, STREAM
  – Stream-in, relation-out: Hancock
Queries - I

• DBS: one-time (transient) queries
• DSMS: continuous (persistent) queries
  – Support persistent and transient queries
  – Predefined and ad hoc queries (CQs)
  – Examples (persistent CQs):
    • Tapestry: content-based email, news filtering
    • OpenCQ, NiagaraCQ: monitor web sites
    • Chronicle: incremental view maintenance

• Unbounded memory requirements
• Blocking operators: window techniques
• Queries referencing past data
Queries - II

• DBS: (mostly) exact query answer
• DSMS: (mostly) approximate query answer
  – Approximate query answers have been studied:
    • Synopsis construction: histograms, sampling, sketches
    • Approximating query answers: using synopsis structures
    • Approximate joins: using windows to limit scope
    • Approximate aggregates: using synopsis structures
• Batch processing
• Data reduction: sampling, synopses, sketches, wavelets, histograms, …
One-pass Query Evaluation

• **DBS:**
  - Arbitrary data access
  - One/few pass algorithms have been studied:
    • Limited memory selection/sorting: \( n \)-pass quantiles
    • Tertiary memory databases: reordering execution
    • Complex aggregates: bounding number of passes

• **DSMS:**
  - Per-element processing: single pass to reduce drops
  - Block processing: multiple passes to optimize I/O cost
Query Plan

- **DBS**: fixed query plans optimized at beginning
- **DSMS**: adaptive query operators
  - Adaptive plans
    - Adaptive query plans have been studied:
      - Query scrambling: wide-area data access
      - Eddies: volatile, unpredictable environments
Query Languages & Processing

• Stream query language issues (compositionality, windows)
• SQL-like proposals suitably extended for a stream environment:
  – Composable SQL operators
  – Queries reference relations or streams
  – Queries produce relations or streams
• Query operators (selection/projection, join, aggregation)
• Examples:
  – GSQL (Gigascope)
  – CQL (STREAM)
• Optimization objectives
• Multi-query execution
Query Languages

3 querying paradigms for streaming data:

1. **Relation-based**: SQL-like syntax and enhanced support for windows and ordering, e.g., CQL (STREAM), StreaQuel (TelegraphCQ), AQuery, GigaScope.

2. **Object-based**: object-oriented stream modeling, classify stream elements according to type hierarchy, e.g., Tribeca, or model the sources as ADTs, e.g., COUGAR.

3. **Procedural**: users specify the data flow, e.g., Aurora, users construct query plans via a graphical interface.

(1) and (2) are declarative query languages, currently, the relation-based paradigm is mostly used.
Windows

- Mechanism for extracting a finite relation from an infinite stream
- Various window proposals for restricting operator scope
  - Windows based on ordering attributes (e.g., time)
  - Windows based on tuple counts
  - Windows based on explicit markers (e.g., punctuations)
  - Variants (e.g., partitioning tuples in a window)
Ordering Attribute Based Windows

- Assumes the existence of an attribute that defines the order of stream elements/tuples (e.g., time)
- Let T be the window length (size) expressed in units of the ordering attribute (e.g., T may be a time window)
- Various possibilities exist:
Tuple Count Based Windows

- Window of size N tuples (sliding, shifting) over the stream
- Problematic with non-unique time stamps associated with tuples
- Ties broken arbitrarily may lead to non-deterministic output
Punctuation Based Windows

• Application inserted “end-of-processing” markers
  – Each data item identifies “beginning-of-processing”
• Enables data item-dependent variable length windows
  – e.g., a stream of auctions
• Similar utility in query processing
  – Limit the scope of query operators relative to the stream
Sample Stream

Traffic (  
  sourceIP -- source IP address
  sourcePort -- port number on source
  destIP -- destination IP address
  destPort -- port number on destination
  length -- length in bytes
  time -- time stamp
);

Selections, Projections

- Selections, (duplicate preserving) projections are straightforward
  - Local, per-element operators
  - Duplicate eliminating projection is like grouping
- Projection needs to include ordering attribute
  - No restriction for position ordered streams

```
SELECT sourceIP, time
FROM Traffic
WHERE length > 512
```
Join Operators

- General case of join operators problematic on streams
  - May need to join arbitrarily far apart stream tuples
  - Equijoin on stream ordering attributes is tractable
- Majority of work focuses on joins between streams with windows specified on each stream

```sql
SELECT A.sourceIP, B.sourceIP
FROM Traffic1 A [window T1], Traffic2 B [window T2]
WHERE A.destIP = B.destIP
```
Aggregation

• General form:
  – `select G, F1 from S where P group by G` having `F2 op 9`
  – G: grouping attributes, F1,F2: aggregate expressions

• Aggregate expressions:
  – distributive: sum, count, min, max
  – algebraic: avg
  – holistic: count-distinct, median
Aggregation in Theory

• An aggregate query result can be streamed if group by attributes include the ordering attribute.

• A single stream aggregate query “select G,F from S where P group by G” can be executed in bounded memory if:
  – every attribute in G is bounded
  – no aggregate expression in F, executed on an unbounded
  – attribute, is holistic

• Conditions for bounded memory execution of aggregate queries on multiple streams
Aggregation & Approximation

• When aggregates cannot be computed exactly in limited storage, approximation may be possible and acceptable

• Examples:
  – select G, median(A) from S group by G
  – select G, count(distinct A) from S group by G
  – select G, count(*) from S group by G having count(*) > f|S|

• Data reduction: use summary structures
  – samples, histograms, sketches …

• Focus of different tutorial
Sampling

- A small random sample $S$ of the data often well-represents all the data
  - Example: select $\text{agg}$ from $R$ where $R.e$ is odd ($n=12$)
    - Data stream: $[9 \ 3 \ 5 \ 2 \ 7 \ 1 \ 6 \ 5 \ 8 \ 4 \ 9 \ 1]$
    - Sample $S$: $[9 \ 5 \ 1 \ 8]$
  - If $\text{agg}$ is $\text{avg}$, return average of odd elements in $S$
    - answer: $5$
  - If $\text{agg}$ is $\text{count}$, return average over all elements $e$ in $S$ of
    - $n$ if $e$ is odd
    - $0$ if $e$ is even
    - answer: $12 \times 3 / 4 = 9$  Unbiased!
Histograms

- Histograms approximate the frequency distribution of element values in a stream
- A histogram (typically) consists of
  - A partitioning of element domain values into buckets
  - A count $C_B$ per bucket $B$ (of the number of elements in $B$)
- Long history of use for selectivity estimation within a query optimizer
Wavelets

- For hierarchical decomposition of functions/signals
- Haar wavelets
  - Simplest wavelet basis => Recursive pairwise averaging and differencing at different resolutions

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Averages</th>
<th>Detail Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>[2, 2, 0, 2, 3, 5, 4, 4]</td>
<td>----</td>
</tr>
<tr>
<td>2</td>
<td>[2, 1, 4, 4]</td>
<td>[0, -1, -1, 0]</td>
</tr>
<tr>
<td>1</td>
<td>[1.5, 4]</td>
<td>[0.5, 0]</td>
</tr>
<tr>
<td>0</td>
<td>[2.75]</td>
<td>[-1.25]</td>
</tr>
</tbody>
</table>

Haar wavelet decomposition: [2.75, -1.25, 0.5, 0, 0, -1, -1, 0]
Query Optimization

- DBS: table based cardinalities used in query optimization => Problematic in a streaming environment
- Cost metrics and statistics: accuracy and reporting delay vs. memory usage, output rate, power usage
- Query optimization: query rewriting to minimize cost metric, adaptive query plans, due to changing processing time of operators, selectivity of predicates, and stream arrival rates
- Query optimization techniques
  - stream rate based
  - resource based
  - QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability
Disorder in Data Streams

• Many queries over data streams rely on some kind of order on the input data items
  – Can often use more efficient operator implementations if the input is sorted on “interesting attributes” (e.g. aggregates)

• What causes disorder in streams?
  – Items from the same source may take different routes
  – Many sources with varying delays
  – May have been sorted on different attribute

• Sorting a stream may be undesirable

• May be more than one possible interesting order over a stream
  – For example, data items may have creation time and arrival time
  – Sorted on arrival time, but creation time also interesting
Punctuations

- Punctuations embedded in stream denote end of subset of data
  - Unblocks blocking operators
  - Reduces state required by stateful operators
- New operator: Punctuate
  - Has special knowledge regarding the input stream
    - timer-based, k-constraints, communication with stream source
  - Emits punctuations in source schema based on special knowledge
- Punctuations can help in two ways:
- Maintain order – Punctuations unblock sort
  - Similar to approach in Gigascope
  - Order-preserving operators include sort behavior for punctuations
- Allow disorder – Punctuations define the end of subsets
  - Operators use punctuations, not order, to output results
  - Reduces tuple latency
IP Network Application: P2P Traffic Detection

- AT&T IP customer wanted to accurately monitor P2P traffic evolution within its network
  - Netflow can be used to determine P2P traffic volumes using TCP port number found in Netflow data
  - P2P traffic might not use known P2P port numbers
  - Using Gigascope SQL-based packet monitor
    - Search for P2P related keywords within each TCP datagram
    - Identified 3 times more traffic as P2P than Netflow

- **Lessons:**
  - Essential to query massive volume data streams
  - Layer independence
  - Correlation of different sources (different app.)
Example - I: Queries for Network Traffic Management

- Large network, e.g., backbone network of ISP
- Monitor a variety of continuous data streams that may be unpredictable and have high data rates
- Provide a "general-purpose" system for monitoring
- Traditional DBS do not support on-line continuous query processing
- Example: network packet traces from multiple network links, here only two specific links: customer link C, backbone link B, we consider only five packet header fields: src, desr, id, len, time
Example - II: Queries for Network Traffic Management

- Compute load on link \( B \) averaged over one-minute intervals, notifying the network operator when the load crosses a specified threshold \( t \).
  Two special functions: getminute, notifyoperator

```
SELECT    notifyoperator(sum(len))
FROM       B
GROUP BY   getminute(time)
HAVING     sum(len) > t
```
Example - III: Queries for Network Traffic Management

- Isolate flows in the backbone link and determine amount of traffic generated by each flow. Flow: sequence of packets grouped in time, and sent from a specific source to a specific destination.

```sql
SELECT flowid, src, dest, sum(len) AS flowlen
FROM (SELECT src, dest, len, time
      FROM B
      ORDER BY time)
GROUP BY src, dest, getflowid(src, dest, time)
AS flowid
```
Example - IV: Queries for Network Traffic Management

• Ad hoc continuous queries when network is congested to determine whether the customer network is the cause.

```
SELECT count(*)
FROM C, B
WHERE C.src = B.src and C.dest = B.dest
and C.id = B.id /
(SELECT count(*) FROM B)
```
Example - V: Queries for Network Traffic Management

• Continuous query for monitoring the source-destination pairs in the top 5% in terms of backbone traffic.

WITH Load AS
  (SELECT src, dest, sum(len) AS traffic
   FROM B
   GROUP BY src, dest)
SELECT src, dest, traffic
FROM Load AS L1
WHERE (SELECT count (*)
   FROM Load AS L2
   WHERE L2.traffic > L1.traffic) >
   (SELECT 0.95xcount(*) FROM Load)
ORDER BY traffic
Query Processing - I

• Continuous query plans:
  – push-based approaches - data is pushed to the DSMS by the source(s)
  – trad.DBS approaches are pull-based, queue problems (overflows)
  – open problems: redesign disk-based data structures and indices

• Processing multiple continuous queries:
  – sharing query plans
  – indexing query predicates

• Distributed query processing:
  – multiple data streams arriving from remote sources
  => distributed optimization strategies
Query Processing - II

(1) Non-blocking operators - 3 techniques for unblocking stream operators:
   • windowing
   • incremental evaluation
   • exploiting stream constraints (punctuations)

(2) Approximate algorithms – if (1) does not work, compact stream summaries may be stored and approximate queries may be run over the summaries
   -> Trade-off: accuracy vs. amount of memory
   Methods of generating synopses: counting methods, hashing methods, sampling methods, sketches, wavelet transformations

(3) Sliding window algorithms:
   • windowed sampling
   • symmetric hash join

(4) On-line data stream mining (single pass): computing stream signatures, decision trees, forecasting, k-medians clustering, nearest neighbour queries, regression analysis, similarity detection, pattern matching
Approximate Query Answering Methods

- **Sliding windows**
  - Only over sliding windows of *recent stream data*
  - Approximation but often more desirable in applications

- **Batched processing, sampling and synopses**
  - **Batched** if update is fast but computing is slow
    - Compute periodically, not very timely
  - **Sampling** if update is slow but computing is fast
    - Compute using sample data, but not good for joins, etc.
  - **Synopsis** data structures
    - Maintain a small *synopsis* or *sketch* of data
    - Good for querying historical data

- **Blocking operators, e.g., sorting, avg, min, etc.**
  - **Blocking** if unable to produce the first output until seeing the entire input
Query Optimization

- DBS: table based cardinalities used in query optimization => Problematic in a streaming environment
- Cost metrics and statistics: accuracy and reporting delay vs. memory usage, output rate, power usage
- Query optimization: query rewriting to minimize cost metric, adaptive query plans, due to changing processing time of operators, selectivity of predicates, and stream arrival rates
- Query optimization techniques
  - stream rate based
  - resource based
  - QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability
Traditional Query Optimization

Statistics Manager:
Periodically collects statistics, e.g., table sizes, histograms

Estimated statistics

Optimizer:
Finds “best” query plan to process this query

Which statistics are required

Executor:
Runs chosen plan to completion

Query

Chosen query plan

[Babu 2004]
STREAM - Optimizing CQs

- Continuous queries are long-running
- Stream characteristics can change over time
  - Data properties: Selectivities, correlations
  - Arrival properties: Bursts, delays
- System conditions can change over time
  ➔ Performance of a fixed plan can change significantly over time
  ➔ Adaptive processing: find best plan for current conditions

[Babu 2004]
STREAM - Traditional Optimization → StreaMon

Profiler:
Monitors current stream and system characteristics

Estimated statistics

Reoptimizer:
Ensures that plan is efficient for current characteristics

Executor:
Executes current plan

Which statistics are required

Decisions to adapt

Combined in part for efficiency

[Babu 2004]
STREAM - Pipelined Filters

- Order commutative filters over a stream
- Example: Track HTTP packets with destination address matching a prefix in given table and content matching "*.ida"
- Simple to complex filters
  - Boolean predicates
  - Table lookups
  - Pattern matching
  - User-defined functions
  - Joins as we will see later

[Babu 2004]
STREAM - Metrics for an Adaptive Algorithm

- Speed of adaptivity
  - Detecting changes and finding new plan

- Run-time overhead
  - Collecting statistics, reoptimization, plan migration

- Convergence properties
  - Plan properties under stable statistics

[Babu 2004]
Optimization Objectives

• Rate-based optimization:
  – Take into account the rates of the streams in the query evaluation tree during optimization
  – Rates can be known and/or estimated

• Maximize tuple output rate for a query
  – Instead of seeking the least cost plan, seek the plan with the highest tuple output rate
Rate Based Optimization – I

- Output rate of a plan: number of tuples produced per unit time
- Derive expressions for the rate of each operator
- Combine expressions to derive expression $r(t)$ for the plan output rate as a function of time:
  - Optimize for a specific point in time in the execution
  - Optimize for the output production size
Rate Based Optimization – II

• Optimize for resource (memory) consumption
• A query plan consists of interacting operators, with each tuple passing through a sequence of operators
• When streams are bursty, tuple backlog between operators may increase, affecting memory requirements
• Goal: scheduling policies that minimize resource consumption
Operator Scheduling

- When tuple arrival rate is uniform:
  - a simple FIFO scheduling policy suffices
  - let each tuple flow through the relevant operators

<table>
<thead>
<tr>
<th>Time</th>
<th>Greedy</th>
<th>FIFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>1.6</td>
<td>2.2</td>
</tr>
<tr>
<td>4</td>
<td>1.8</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Average arrival rate: 0.5 tuples/sec

FIFO: tuples processed in arrival order

Greedy: if tuple before s1 schedule it; otherwise process tuples before s2
Progress Chart: Chain Scheduling

- assign priorities to operators equal to the slope of the lower envelope segment to which the operator belongs
- Schedule the operator with the highest priority
QoS Based Optimization

• Query and operator scheduling based on QoS requirements
• Two-level scheduling policy:
  – Operator batching: superbox selection, superbox traversal based on avg throughput, avg latency, minimizing memory
  – Tuple batching

Optimization Objectives
• Multi-way join techniques proposed:
  – start with a fixed plan
  – moderately adjust it as tuples arrive
• Eddies framework for adaptive query optimization:
  – Continuously adapt the evaluation order as tuples arrive
Load Shedding

• When input stream rate exceeds system capacity a stream manager can shed load (tuples)
• Load shedding affects queries and their answers
• Introducing load shedding in a data stream manager is a challenging problem
• Random and semantic load shedding
Load Shedding in Aurora

• QoS for each application as a function relating output to its utility
  – Delay based, drop based, value based

• Techniques for introducing load shedding operators in a plan such that QoS is disrupted the least
  – Determining when, where and how much load to shed
Load Shedding in STREAM

• Formulate load shedding as an optimization problem for multiple sliding window aggregate queries
  – Minimize inaccuracy in answers subject to output rate matching or exceeding arrival rate

• Consider placement of load shedding operators in query plan
  – Each operator sheds load uniformly with probability $p_i$
Multi-query Processing

• In traditional multi-query optimization:
  – sharing (of expressions, results, etc.) among queries can lead
  – to improved performance

• Similar issues arise when processing queries on streams:
  – sharing between select/project expressions
  – sharing between sliding window join expressions
Grouped Filters

<table>
<thead>
<tr>
<th>Select Predicates for Stream S.A</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.A &gt; 1</td>
</tr>
<tr>
<td>S.A &gt; 7</td>
</tr>
<tr>
<td>S.A &gt; 11</td>
</tr>
<tr>
<td>S.A &lt; 3</td>
</tr>
<tr>
<td>S.A &lt; 5</td>
</tr>
<tr>
<td>S.A = 6</td>
</tr>
<tr>
<td>S.A = 8</td>
</tr>
</tbody>
</table>

Tuple S.A = 8
Shared Window Joins

Consider the two queries:

```
select sum (A.length
from   Traffic1 A [window 1 hour],
       Traffic2 B [window 1 hour]
where A.destIP = B.destIP
```

```
select count (distinct A.sourceIP)
from   Traffic1 A [window 1 min],
       Traffic2 B [window 1 min]
where A.destIP = B.destIP
```

- Great opportunity for optimization as windows are highly shared
- Strategies for scheduling the evaluation of shared joins:
  - Largest window only
  - Smallest window first
  - Process at any instant the tuple that is likely to benefit the largest number of joins (maximize throughput)
Stream Data Mining

• Stream mining
  – It shares most of the difficulties with stream querying
  – Patterns are hidden and more general than querying
  – It may require exploratory analysis
    • Not necessarily continuous queries

• Stream data mining tasks
  – Multi-dimensional on-line analysis of streams
  – Mining outliers and unusual patterns in stream data
  – Clustering data streams
  – Classification of stream data
Stream Mining - Challenges

• Most stream data are at pretty low-level or multi-dimensional in nature: needs ML/MD processing

• Analysis requirements
  – Multi-dimensional trends and unusual patterns
  – Capturing important changes at multi-dimensions/levels
  – Fast, real-time detection and response
  – Comparing with data cube: Similarity and differences

• Stream (data) cube or stream OLAP: Is this feasible?
  – Can we implement it efficiently?

[Han 2004]
Examples: Multi-Dimensional Stream Analysis

• Analysis of Web click streams
  – Raw data at low levels: seconds, web page addresses, user IP addresses, …
  – Analysts want: changes, trends, unusual patterns, at reasonable levels of details
  – E.g., *Average clicking traffic in North America on sports in the last 15 minutes is 40% higher than that in the last 24 hours.*

• Analysis of power consumption streams
  – Raw data: power consumption flow for every household, every minute
  – Patterns one may find: *average hourly power consumption surges up 30% for manufacturing companies in Chicago in the last 2 hours today than that of the same day a week ago*
Stream Data Reduction

Challenges of OLAPing stream data
- Raw data cannot be stored
- Simple aggregates are not powerful enough
- History shape and patterns at different levels are desirable: multi-dimensional regression analysis

Proposal
- A scalable multi-dimensional stream “data cube” that can aggregate regression model of stream data efficiently without accessing the raw data

Stream data compression
- Compress the stream data to support memory- and time-efficient multi-dimensional regression analysis

[Han 2004]
Data Warehouse: Stream Cube Architecture

- A tilt time frame
  - Different time granularities (second, minute, quarter, hour, day, week, ...)
- Critical layers
  - Minimum interest layer (m-layer)
  - Observation layer (o-layer)
  - User: watches at o-layer and occasionally needs to drill-down down to m-layer
- Partial materialization of stream cubes
  - Full materialization: too space and time consuming
  - No materialization: slow response at query time
  - Partial materialization: what do we mean “partial”?
- On-line materialization
  - Materialization takes precious resources and time
    - Only incremental materialization (with slide window)
  - Only materialize “cuboids” of the critical layers?
    - Some intermediate cells that should be materialized
  - Popular path approach vs. exception cell approach
    - Materialize intermediate cells along the popular paths
    - Exception cells: how to set up exception thresholds?
    - Notice exceptions do not have monotonic behaviour
- Computation problem
  - How to compute and store stream cubes efficiently?
  - How to discover unusual cells between the critical layer?
Data Warehouse: Stream Cube Computation

• Cube structure from m-layer to o-layer
• Three approaches
  – All cuboids approach
    • Materializing all cells (too much in both space and time)
  – Exceptional cells approach
    • Materializing only exceptional cells (saves space but not time to compute and definition of exception is not flexible)
  – Popular path approach
    • Computing and materializing cells only along a popular path
    • Using H-tree structure to store computed cells (which form the stream cube—a selectively materialized cube)

[Han 2004]
Other Approaches for Mining Unusual Patterns in Stream Data

• Beyond multi-dimensional regression analysis
  – Other approaches can be effective for mining unusual patterns

• Multi-dimensional gradient analysis of multiple streams
  – Gradient analysis: finding substantial changes (notable gradients) in relevance to other dimensions
  – E.g., those stocks that increase over 10% in the last hour

• Clustering and outlier analysis for stream mining
  – Clustering data streams
  – History-sensitive, high-quality incremental clustering

• Decision tree analysis of stream data
  – Evolution of decision trees
  – Incremental integration of new streams in decision-tree induction

[Han 2004]
Research Problems: Stream Classification

- What about decision tree may need dramatic restructuring?
  - Especially when new data is rather different from the existing model
  - Efficient detection of outliers (far away from majority) using constructed models
- Weighted by history of the data: pay more attention to new data?
- Mining evolutions and changes of models?
- Multi-dimensional decision tree analysis?
- Stream classification with other classification approaches?
- Constraint-based classification with data streams?

[Han 2004]
Research Problems: Stream Data Mining

• Stream data mining: should it be a general approach or application-specific ones?
  – Do stream mining applications share common core requirements and features?
• Killer applications in stream data mining
• General architectures and mining language
• Multi-dimensional, multi-level stream data mining
  – Algorithms and applications
• How will stream mining make good use of user-specified constraints?
• Stream association and correlation analysis
  – Measures: approximation? Without seeing the global picture?
  – How to mine changes of associations?
Outline

• Introduction:
  – What are DSMS? (terms)
  – Why do we need DSMS? (applications)
• Example 1:
  – Network monitoring with TelegraphCQ
• Concepts and issues:
  – Architecture(s)
  – Data modeling
  – Query processing and optimization
  – Data reduction
  – Stream Mining
• Overview of existing systems
• Example 2:
  – DSMS for sensor networks
• Summary:
  – Open issues
  – Conclusions
Systems

• **Aurora** (Brandeis, Brown, MIT, [http://www.cs.brown.edu/research/aurora](http://www.cs.brown.edu/research/aurora)): workflow-oriented system, sensor monitoring, dataflow

• **COUGAR** (Cornell, [http://www.cs.cornell.edu/database/cougar](http://www.cs.cornell.edu/database/cougar)): sensor database, time series

• **GigaScope** (AT&T): distributed network monitoring architecture, proprietary system

• **Hancock** (AT&T): telecom streams

• **NiagaraCQ** (OGI/Wisconsin, [http://www.cs.wisc.edu/niagara](http://www.cs.wisc.edu/niagara)): continuous XML query system for dynamic web content

• **OpenCQ** (Georgia Tech, [http://disl.cc.gatech.edu/CQ](http://disl.cc.gatech.edu/CQ)): continuous query system for monitoring streaming web content, triggers, incr. view maintenance

• **StatStream** ([http://cs.nyu.edu/cs/faculty/shasha/papers/statstream.html](http://cs.nyu.edu/cs/faculty/shasha/papers/statstream.html)): multi-stream monitoring system for on-line statistics

• **STREAM** (Stanford, [http://www-db.stanford.edu/stream](http://www-db.stanford.edu/stream)): general-purpose relation-based system

• **Streaminer** (UIUC): stream data mining project

• **Tapestry** (Xerox): pub/sub content-based filtering

• **TelegraphCQ** (UC Berkeley, [http://telegraph.cs.berkeley.edu](http://telegraph.cs.berkeley.edu)): adaptive engine for sensors, continuous query processing system

• **Tradebot** ([www.tradebot.com](http://www.tradebot.com)): stock tickers & streams

• **Tribeca** (Bellcore): network monitoring, early on-line Internet traffic monitoring tool
Aurora

• Data processing system targeted towards monitoring applications:
  – Streams: for each monitoring task DBA adds 1-n triggers into trigger network
  – Large network of triggers
  – Imprecise data
  – Real-time requirements
• Specified set of operators, connected in a data flow graph
• Each trigger is data flow graph (each node is one of seven built-in operators)
• Optimization of:
  – Data flow graph
  – Compile-time and run-time optimization of trigger network
• Three query modes (continuous, ad-hoc, view)
• Detects resource overload: accepts QoS specifications and attempts to optimize QoS for outputs produced
• Real-time scheduling, introspection and load shedding
GigaScope

- Specialized stream database for network applications
- GSQL for declarative query specifications: pure stream query language (stream input/output)
- Uses ordering attributes in IP streams (timestamps and their properties) to turn blocking operators into non blocking ones
- GSQL processor is code generator.
- Query optimization uses a two level hierarchy
Hancock

- A C-based domain specific language which facilitates transactor signature extraction from transactional data streams
- Support for efficient and tunable representation of signature collections
- Support for custom scalable persistent data structures
- Elaborate statistics collection from streams
NiagaraCQ

- CQs for monitoring persistent data sets distributed over WAN
- Scalability (# queries) by grouping CQs for efficient evaluation
- Problem of blocking operators in query plans for streams
OpenCQ

• CQs for monitoring persistent data sets distributed over WAN
• QP based on incremental view maintenance
STREAM

• General purpose stream data manager
  – Data streams and stored relations
• CQL (continuous query language) for declarative query specification
• Timestamps in streams
• Flexible query plan generation
• Query processing architecture
• Resource management:
  – Operator scheduling
  – Graceful approximation: can handle high data rates
• Static and dynamic approximations
Tapestry

• CQs for content-based filtering
  – Over append-only database containing email and bulletin board messages

• Restricted subset of SQL
  – To guarantee efficient evaluation and append-only results
Telegraph

- CQ processing system
  - Uses adaptive query engine
  - Query execution strategies over data streams generated by sensors
  - Processing techniques for multiple CQs
- Support for stream oriented operators
- Support for adaptivity in query processing
  - Optimization
- Various aspects of optimized multi-query stream processing
Tribeca

- Restricted querying capability over network packet streams
# System Comparison

<table>
<thead>
<tr>
<th>System</th>
<th>Data Stream Architecture</th>
<th>Data Model</th>
<th>Query Language</th>
<th>Query Answers</th>
<th>Query Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aurora</td>
<td>low-level</td>
<td>RS-in RS-out</td>
<td>Operators</td>
<td>approximate</td>
<td>QoS-based, load shedding</td>
</tr>
<tr>
<td>Gigascope</td>
<td>two level (low, high)</td>
<td>S-in S-out</td>
<td>GSQL</td>
<td>exact</td>
<td>decomposition, avoid drops</td>
</tr>
<tr>
<td>Hancock</td>
<td>High-level</td>
<td>RS-in R-out</td>
<td>Procedural</td>
<td>exact, signatures</td>
<td>optimize for I/O, process blocks</td>
</tr>
<tr>
<td>STREAM</td>
<td>low-level</td>
<td>RS-in RS-out</td>
<td>CQL</td>
<td>approximate</td>
<td>optimize space, static analysis</td>
</tr>
<tr>
<td>Telegraph</td>
<td>high-level</td>
<td>RS-in RS-out</td>
<td>SQL-based</td>
<td>exact</td>
<td>adaptive plans, multi-query</td>
</tr>
</tbody>
</table>
Example 1: Traffic Analysis

- Need to analyze Internet traffic is increasing ....
- .... and so is the number of tools for this
- Examples:
  - ISP monitor service levels, look for bottlenecks, etc.
  - development of new protocols, like P2P
- Basic structure of tools:
Traffic Analysis (cont.)

- Performing traffic analysis to gain new knowledge is an iterative process:
  - Packet capturing
  - Analysis
  - Develop new analysis
  - New insights
  - Network link
Expectations

• Be helpful for typical traffic analysis tasks:
  – the load of a system
    • how often are certain ports, like FTP, or HTTP, of a server contacted
    • which share of bandwidth is used by different applications
    • which departments use how much bandwidth on the university backbone
  – characteristics of flows
    • distribution of life time and size of flows
    • relation between number of lost packets and life time of flows
    • what are the reasons for throughput limitations, or
  – characteristics of sessions:
    • how long do clients interact with a web server
    • which response time do clients accept from servers
    • how long are P2P clients on-line after they have successfully downloaded a file
Expectations (cont.)

- Allow online and offline analysis
- Manage data and analyze data with the same tool
- Facilitate development and reuse of analysis components
Expectations (cont.)

• Provide sufficient performance:
  – idealized gigabit/s link
    • all packets 1500 byte, TCP/IP header 64 byte
    • 42 megabit/s of header information
  – more realistic: compression of 9:1 or less
    • approx. 880 megabit/s on gigabit/s link
    • approx. 11 megabit/s for 100 megabit/s network
Approach

• Public domain DSMS (fall 2003):
  – TelegraphCQ
  – Aurora ... only source tree, complete??

• Student project by A. Bergamini & G. Tulo:
  – install TelegraphCQ
  – connect it to wrappers, i.e., sources
  – model TCP traces/streams
  – develop queries for simple but typical tasks
  – try to re-implement an existing complex tool
  – identify performance bounds
TelegraphCQ

• Characterization of it’s developers:
  – “a system for continuous dataflow processing”
  – “aims at handling large streams of continuous queries over high-volume highly variable data streams”

• Extends PostgreSQL
  – adaptive query processing operators
  – shared continuous queries
  – data ingress operations
TelegraphCQ Architecture

Phase 1: Data Acquisition
- Source1 TCPdump
- TCQ Wrapper
- Source2 TCPdump
- TCQ Wrapper

Shared Memory Infrastructure
- TCQ Clearing House
- TCQ BackEnd
- TCQ FrontEnd

Phase 2: Continuous Query Execution

Phase 3: Presentation of results

Client
Continuous Queries in TCQ

• Data streams are defined in DDL with CREATE STREAM (like tables)

```
SELECT  <select_list>
FROM    <relation_and_pstream_list>
WHERE   <predicate>
GROUP BY <group_by_expressions>
WINDOW  stream[interval], ...
ORDER BY <order_by_expressions>
```
Continous Queries in TCQ (cont.)

• Restrictions in TelegraphCQ 0.2 alpha release [9]:
  – windows can only be defined over streams (not for PostgreSQL tables)
  – **WHERE** clause qualifications that join two streams may only involve attributes, not attribute expressions or functions
  – **WHERE** clause qualifications that filter tuples must be of the form attribute operand constant
  – **WHERE** clause may only contain **AND** (not **OR**); sub queries are not allowed
  – **GROUP BY** and **ORDER BY** clauses are only allowed in window queries
### Stream Definition

- CREATE STREAM p6trace.tcp (ip_src cidr, ip_dst cidr, hlen bigint, tos int, length bigint, id bigint, frag_off bigint, ttl bigint, prot int, ip_hcsum bigint, port_src bigint, port_dst bigint, sqn bigint, ack bigint, tcp_hlen bigint, flags varchar(10), window bigint, tcp_csum bigint, tcqtime timestamp TIMESTAMP COLUMN) type ARCHIVED;
Task 1

• How many packets have been sent during the last five minutes to certain ports?
• Store all ports of interests in a table and join with the stream
  CREATE TABLE services (port bigint, counter bigint);
  SELECT services.port, count(*)
  FROM p6trace.tcp, services
  WHERE p6trace.tcp.port_dst=services.port
  GROUP BY services.port
  WINDOW p6trace.tcp ['5 min'];
Task 2

- How many bytes have been exchanged on each connection during the last minute?
- Simple heuristic to identify connections:
  - during a one minute window all packets with the same sender and receiver IP addresses and port numbers belong to the same connection

SELECT ip_src, port_src, ip_dst, port_dst, sum(length-ip_len-tcphlen) FROM p6trace.tcp GROUP BY ip_src, port_src, ip_dst, port_dst WINDOW p6trace.tcp [‘1 min’];
Task 3

• How many bytes are exchanged over the different connections during each week?

• Two problems to handle this in a CQ:
  – GROUP BY clause can only be used together with a WINDOW clause
    • window smaller than one week
    • payload of each packet would contribute several times to intermediate results
    • how to remove this redundancy?
    • tumbling or jumping windows are needed
  – identification of connections
    • simple heuristic from task 2 does not work
    • boils down to the generic problem of association identification
Identification of Associations

• Use address fields and rules

• Example: TCP connections
  – \texttt{GROUP BY} addresses only
  – \texttt{rule: if } t_n - t_1 < T \texttt{ then same connection}
  \texttt{else new connection}
Identification of Associations (cont.)

A priori no address values are known

Check for each new packet:
- is address combination known?
  NO: insert new entry
  YES: is it a new or old connection?
    OLD: update statistics
    NEW: insert new connection

<table>
<thead>
<tr>
<th>IP d.</th>
<th>IP s.</th>
<th>Port d.</th>
<th>Port s.</th>
<th>Statistics</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>( t_1 )</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>( t_2 )</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>( t_n )</td>
</tr>
</tbody>
</table>
Identification of Associations (cont.)

A priori no address values are known

Check for each new packet:
-is address combination known?
  NO: insert new entry
  YES: is it a new or old connection?
    OLD: update statistics
    NEW: insert new connection

With a single pass over the data this is only possible with sub-queries in SQL
Task 4

- Which department has used how much bandwidth on the university backbone in the last five minutes?
- Store address ranges of all departments in a table
- Check with “>>” which address range contains the IP address of the packet in the data stream
- CREATE TABLE departments (name varchar(30), prefix cidr, traffic bigint);

SELECT departments.name, sum(length-hlen-tcp_hlen) FROM p6trace.tcp, departments WHERE departments.prefix >> p6trace.tcp.ip_src GROUP BY departments.name WINDOW p6trace.tcp ['5 min'];

- TelegraphCQ prototype produces incorrect results if “>>” is used in a join, but works correctly with “=”
Task 4 (cont.)

• "Solution": use "=" and enumerate all addresses in a stored table
• CREATE TABLE departments (name varchar(30), ip_addr cidr, traffic bigint);

SELECT departments.name, sum(length-hlen-tcp_hlen)
FROM p6trace.tcp, departments
WHERE departments.ip_addr = p6trace.tcp.ip_src
GROUP BY departments.name
WINDOW p6trace.tcp ['5 min'];