Data Stream Management Systems
- for Sensor Networks –

Vera Goebel
Department of Informatics, University of Oslo

INF5100, Fall 2010

• What are DSMSs? (terms)
• Why do we need DSMSs? (applications)
• Concepts: Data Model, Query Processing, Windows
• Application Example: Medical Data Analysis with Esper
Handle Data Streams in DBS?

**Traditional DBS**

- SQL Query
- Query Processing
- Main Memory
- Disk
- Result

**DSMS**

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

<table>
<thead>
<tr>
<th><strong>Database Systems (DBS)</strong></th>
<th><strong>DSMS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent relations (relatively static, stored)</td>
<td>Transient streams (on-line analysis)</td>
</tr>
<tr>
<td>One-time queries</td>
<td>Continuous queries (CQs)</td>
</tr>
<tr>
<td>Random access</td>
<td>Sequential access</td>
</tr>
<tr>
<td>“Unbounded” disk store</td>
<td>Bounded main memory</td>
</tr>
<tr>
<td>Only current state matters</td>
<td>Historical data is important</td>
</tr>
<tr>
<td>No real-time services</td>
<td>Real-time requirements</td>
</tr>
<tr>
<td>Relatively low update rate</td>
<td>Possibly multi-GB arrival rate</td>
</tr>
<tr>
<td>Data at any granularity</td>
<td>Data at fine granularity</td>
</tr>
<tr>
<td>Assume precise data</td>
<td>Data stale/imprecise</td>
</tr>
<tr>
<td>Access plan determined by query processor, physical DB design</td>
<td>Unpredictable/variable data arrival and characteristics</td>
</tr>
</tbody>
</table>

Adapted from [Motawani: PODS tutorial]
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, …
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

• Near real-time queries/analyses
  – ISPs: controlling the service level
  – NOAA: tornado detection using weather radar data
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><strong>Disc Drive Performance</strong></td>
<td><strong>60 msec</strong></td>
<td><strong>5.3 msec</strong></td>
<td><strong>11 x</strong></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.)

• 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

Streams of Inputs

- **Input Monitor**
- **Working Storage**
- **Summary Storage**
- **Static Storage**

- Updates to Static Data
- User Queries

Streams of Outputs

- **Query Processor**
- **Query Repository**
- **Output Buffer**

- Streaming Inputs
- Streaming Outputs

**Updates to Static Data**

- **User Queries**

---

[Golab & Özsu 2003]
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
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
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
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., Esper, 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

VLDB 2003 Tutorial [Koudas & Srivastava 2003]
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

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 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 agg is avg, return average of odd elements in S
    - answer: 5
  - If agg is count, return average over all elements e in S of
    - n if e is odd
    - 0 if e is even
    - answer: 12*3/4 = 9

Unbiased!
Histories

- 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
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

[Han 2004]
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

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

**Executor:** Runs chosen plan to completion

Which statistics are required
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

Which statistics are required

Profiler:
Monitors current stream and system characteristics

Estimated statistics

Reoptimizer:
Ensures that plan is efficient for current characteristics

Decisions to adapt

Combined in part for efficiency

Executor:
Executes current plan

Query
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

\[ \sigma_1 \sigma_2 \sigma_3 \]

[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

[Profiler] → [Reoptimizer] → [Executor]

StreaMon

[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
Some Sensornet Applications

ZebraNet

Redwood forest microclimate monitoring

Smart cooling in data centers

http://www.hpl.hp.com/research/dca/smart_cooling/
Application Examples

Habitat Monitoring:
Storm petrels on Great Duck Island, microclimates on James Reserve.

Vehicle detection: sensors along a road, collect data about passing vehicles.

Earthquake monitoring in shake-test sites.

Traditional monitoring apparatus.
Sensor Networks

Base station (gateway)

Motes (sensors)
Sensor Network Characteristics

• Autonomous nodes
  – Small, low-cost, low-power, multifunctional
  – Sensing, data processing, and communicating components

• Sensor network is composed of large number of sensor nodes
  – Proximity to physical phenomena
    • Deployed inside the phenomenon or very close to it

• Monitoring and collecting physical data

• No human interaction for weeks or months at a time
  – Long-term, low-power nature
Sensor Hardware

- A sensor node is made up of four basic components
  - Sensing unit
    - usually composed of two subunits: sensors and analog to digital converters (ADCs).
  - Processing unit,
    - Manages the procedures that make the sensor node collaborate with the other nodes to carry out the assigned sensing tasks.
  - Transceiver unit
    - Connects the node to the network.
  - Power units (the most important unit)
- Matchbox-sized module
  - consume extremely low power,
  - operate in high volumetric densities,
  - have low production cost and be dispensable,
  - be autonomous and operate unattended,
  - be adaptive to the environment.
Principles of Sensor Networks

- A large number of low-cost, low-power, multifunctional, and small sensor nodes
- Sensor node consists of sensing, data processing, and communicating components
- A sensor network is composed of a large number of sensor nodes,
  - which are densely deployed either inside the phenomenon or very close to it.
- The position of sensor nodes need not be engineered or pre-determined.
  - sensor network protocols and algorithms must possess self-organizing capabilities.
Managing Data

• **Purpose of sensor network:**
  
  *Obtain real-world data*
  
  – Extract and combine data from the network

• **But:** Programming sensor networks is hard!
  
  – Months of lifetime required from small batteries
  
  – Lossy, low-bandwidth, short range communication
  
  – Highly distributed environment
  
  – Application development
  
  – Application deployment administration
Data Management Systems for Sensor Networks

• Motivation:
  – Implement data access
    • Sensor tasking
    • Data processing
    • Possibly support for data model and query language

• Goals:
  – Adaptive
    • Network conditions
    • Varying/unplanned stimuli
  – Energy efficient
    • In-network processing
    • Flexible tasking
    • Duty cycling
DSMS for Sensor Networks

• Aurora & Medusa System
  – Aurora: single-site high performance stream processing engine
  – Aurora*: connecting multiple Aurora workflows in a distributed environment
  – Medusa: distributed environment where hosts belong to different organizations and no common QoS notion is feasible

• TinyDB
  – Developed as public-domain system at UC Berkeley
  – Widely used by research groups as well as industry pilot projects
  – Successful deployments in Intel Berkeley Lab and redwood trees at UC Botanical Garden
Health Care Applications

• Integrated patient monitoring
• Telemonitoring of human physiological data
• Tracking and monitoring doctors and patients inside a hospital
• Tracking and monitoring patients and rescue personnel during rescue operations
Online Analysis of Myocardial Ischemia From Medical Sensor Data Streams with Esper

Stig Støa¹, Morten Lindeberg², Vera Goebel²

¹ The Interventional Centre (IVS), Rikshospitalet University Hospital, Oslo, Norway
² Distributed Multimedia Systems, Department of Informatics, University of Oslo, Norway
Adaptive Sized Windows To Improve Real-Time Health Monitoring – A Case Study on Heart Attack Prediction

- **Application**: Real-time health monitoring.
- **Problem**: Data stream management systems (DSMSs) mainly support the processing of data stream windows of static size. Should adapt to the physiological processes of the human body, e.g., the cardiac cycle, which has variable durations.
- **Goal**: Adapt the processing of data streams to physiological processes, such as heartbeats, using *time-based sliding windows of adaptive “size.”*

- Published work in biomedical symposium:
  Stig Støa, Morten Lindeberg, Vera Goebel: Online Analysis of Myocardial Ischemia from Medical Sensor Data Streams with Esper, *Proceedings of the First International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL 2008), October 2008*
Idea

- Let external events (tuple results from external query) determine the window size of a sliding window
- ECG stream to detect heartbeats (*QRS detection*)
- Accelerometer stream to detect heart displacement (*Ischemia detection*)
- Output of *QRS detection* (delay) determines when to trigger the flushing of the sliding window in *Ischemia detection* query
- ‘Delay’ is used to slow down accelerometer stream to account for QRS detection delay in the *FIFO queue*
Experiment Goal #1

- Recreate off-line technique (Elle et al. 2005) conducted in MATLAB
- Early recognition of regional cardiac ischemia
- 3-way accelerometer placed on left ventricle of the heart

**Single metric:**
- Fast Fourier Transformation (FFT) is used to examine the accelerometer signal in the frequency-domain
- Euclidian distance vector ($EDV(i)$) between reference vector $RV(0)$ and current vector $CV(j)$, where $j$ is the latest sample number
  - $CV(j)$ : FFT over sliding window (size 512 over $y$-axis)
  - $RV(0)$ : FFT over baseline window (first 512 samples)

- Data set from surgery performed on pigs at the Interventional Centre
- We can conduct experiments with the same data set (data set 1)
Experiment Goal #2

• Improve results by adding beat-to-beat detection using a QRS detection algorithm on ECG signals
  – Each ECG trace of a normal heartbeat typically contains a QRS event
  – A good reference for separating heartbeats
• We need to perform FFT over sliding windows of variable size!
• Cannot use the same data, use new data set that include ECG (data set 2)
Challenges

1. Incorporate signal processing operations
   - **Problem**: Not supported in the query language
   - Fast Fourier Transformation of the accelerometer signals
   - Euclidian distance vector from baseline window
   - QRS detection for detecting the heartbeats from the ECG signals
   - **Solution**: Custom aggregate functions

2. Static sized windows are not feasible for beat-to-beat detection
   - **Problem**: Heartbeat duration is not a static pre-known size. DSMS window techniques only describe static time-based or tuple-based windows.
   - **Solution**: Introduce variable length triggered tumbling windows

3. Synchronize the two streams
   - **Problem**: QRS detection introduces variable delay (approx. 91 samples)
   - **Solution**: Introduce variable buffer, that “slows” down the accelerometer stream
Signal processing operations

• Implement as custom aggregate functions
• Use defined Java interface and simply add to query engine
• Implemented methods:
  – **QRSD**(v): QRS detection based on algorithm from Hamilton et al. 1986, source code is public available
  – **edv**(v): Euclidian distance from baseline
Variable length triggered tumbling windows

- The ECG stream is aggregated into a stream consisting of QRS events $S_b$.
- This stream ($S_b$) triggers the flushing of the sliding window $w(t)$ where the custom aggregation over the stream $S_a$ is performed.
- This window technique is not supported by Esper => We implemented a “workaround” exploiting functionality of externally timed windows.
Stream Synchronization

- The QRS detection algorithm over the ECG stream introduces a variable delay $\Delta t$.
- Introduce the same delay to the accelerometer stream.
- Accelerometer stream is sent through a FIFO queue with dynamic size.
- QRS detection function sets the dynamic size of the FIFO queue (also triggers the flushing of the aggregate window, in order to obtain dynamic windows).
Results #1 (data set 1)

Occlusion occurs after 80 seconds

Perfusion after 170 seconds

Figure shows a perfect overlap, the technique by Elle et al. 2005 can be recreated online using Esper

Easier than MATLAB

SELECT edv(y)
FROM Accelerometer
WINDOW LENGTH(512)
Results #2 (data set 2)

Plot shows fixed sliding window (512 samples) and dynamic triggered window (based on QRS detection) => less variance!

SELECT edv(y)
FROM Accelerometer
TRIGGER WINDOW BY QRSD(ECG.value)

Sudden drop caused by ultra sound probe

Occlusion

Perfusio

Euclidean distance

Time in milli seconds

0 1 2 3 4 5

−20 −10 0 10 20 30

x 10^5
Results #3 (data set 2)

Query with added local minimum value => easy to change!

SELECT edv(y), min(y) FROM Accelerometer TRIGGER WINDOW BY QRSD(ECG.value)

The bottom plot represents local minimum value for the accelerometer stream
Implementation

- Java and Esper (open source component for event processing available at http://esper.codehaus.org/)
- Use existing window model, Esper is not changed
- Base window boundaries on the manipulated timestamps (registered as external timestamps in the Esper query) calculated from external / trigger query
Case study 1

- Ischemia detection (joint work with IVS, Oslo, Norway)
  - Real data from surgeries on pigs
  - **Accelerometer** attached to heart surface, used to identify irregular movements
  - **ECG** stream is used to detect each heartbeat (QRS Detection)
  - Upon detecting heartbeats, flush current window over the accelerometer stream

# Main query (Ischemia detection):
select edv(y), max(timestamp), max(realtime) from Accelerometer.win:ext_timed(timestamp, 5 second)

# Trigger query (QRS detection):
select qrsdetect(value) from ECG.win:length(1)
Case study 2

- Simple sine signal (we know ground truth)
  - Investigate more thoroughly the effect (overhead) of the window model itself

```sql
# Main query (Average value):
select avg(val), max(origin), max(fakeorigin) from SineTriggerTuple.win:ext_timed(fakeorigin, 1 sec)

# Trigger query (Sine phase detection):
select max(origin), sinephasedetect(val) from SineTriggerTuple.win:length(2)
```
Results

- Improvement of analysis results

Case study 1: Ischemia Detection

Case study 2: Simple sine signal
Results

- Low overhead for memory and CPU of the adaptive window technique confirmed by performance evaluation.
Conclusion

• DSMSs can be used for real-time analysis => easy for medical practitioners to investigate novel methods

• Illustrated a method of online analysis of medical sensor data focusing on detection of myocardial ischemia

• Added beat-to-beat detection by using ECG – Results with less variance

• Introduced a new type of window for DSMSs: *Variable length triggered tumbling windows*