Data Stream Management Systems
- for Sensor Networks –

Vera Goebel
Department of Informatics, University of Oslo

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
- Result

DSMS

- Query Processing
- Main Memory
- Data Stream(s)

Register CQs
Result (stored)

Data Stream(s)

Main Memory

Archive

Scratch store
(main memory or disk)

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>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.)

- 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

Input Monitor

Streaming Inputs

Working Storage Summary Storage Static Storage Query Repository

Query Processor

Output Buffer

Streaming Outputs

Updates to Static Data

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

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 instantiations 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?
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 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.
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

Some Sensornet Applications

Redwood forest microclimate monitoring

Smart cooling in data centers

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

Application Examples

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

Earthquake monitoring in shake-test sites.

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

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:

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(i), where i is the latest sample number
  - CV(i): 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
   - Euclidean 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): Euclidean 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)

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

Results #2 (data set 2)

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

Results #3 (data set 2)

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

Case study 2

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

Results

• Improvement of analysis 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*