Data Stream Management Systems

Tore Risch
Dept. of information technology
Uppsala University
Sweden
2015-12-10
New applications

- Data comes as huge data streams, e.g.
  - Satellite data
  - Scientific instruments
  - Colliders
  - Industrial equipment
  - Patient monitoring
  - Stock data
  - Traffic monitoring
Too much data to store on disk

⇒ Would like to query data directly in the streams
DataBase Management Systems, DBMS

e.g. Oracle, MySQL, SQL Server
DataStream Management Systems, DSMS

e.g. Tipco Streambase

![Diagram of DSMS components]

- Dynamic Continuous Queries (CQs)
  - Query Processor
  - Stream Manager
  - Meta-data
  - Data streams to/from RDB
Data managers, key/value stores

e.g. BerkeleyDB, Riak, Cassandra, DynamoDB
Data Stream Managers

e.g. Storm, Kinesis, Stream Mill

Java/C++/JavaScript programs

Data stream manager

Data streams

Data streams
sa.engine Stream Analytics Engine
Continuous queries over streams from expressways

Schema for stream CarLocStr of tuples:

```sql
CarLocStr(car_id, /* unique car identifier */
speed, /* speed of the car */
exp_way, /* expressway: 0..10 */
lane, /* lane: 0,1,2,3 */
dir, /* direction: 0(east), 1(west) */
x-pos); /* coordinate in express way */
```

Continuous query expressed in the query language CQL (Stanford):

Continuously get the cars in a window of the stream every 30 seconds:

```sql
SELECT DISTINCT car_id
FROM CarLocStr [RANGE 30 SECONDS];
```

Get the average speed of vehicles per expressway, direction, segment each 5 minutes:

```sql
SELECT exp_way, dir, seg, AVG(speed) as speed,
FROM CarSegStr [RANGE 5 MINUTES]
GROUP BY exp_way, dir, seg;
```
DSMSs vs. Relational Databases

- Streams potentially **infinite** in size
  - Regular DBs based on queries to **finite** tables
  - A join (e.g. nested-loop join) may scan table several times
    - NLJ impossible over streams
    - Merge Join (MJ) possible in stream order
  - Cannot materialize (store) intermediate results
    - Regular hash join impossible over streams

- **Streams ordered**
  - Regular relational DBs are based on *sets* and *bags*
  - The order of the result of a relational (sub-query) is insignificant
    - Used in lots of optimization methods
  - Important to **preserve order** when result is stream
DSMSs vs. relational databases

- Often **very high** stream data volume and **rate**
  - Regular DBs much less demanding
  - Regular database are optimized for *high-watermark*
    - Once loaded it does not change in size very fast
  - Streams may have **extreme insert** and **delete** rates as tuples flow through the DSMS

- Real-time delivery, Quality of Service
  - Regular databases also have QoS requirements, but slower!
    - E.g. guaranteed **average** response time of 1s (soft real-time)
  - If a DSMS does not keep up with the flow some actions have to be made
    - E.g. skipping elements, *shedding*
DSMSs vs. relational databases

- Stream queries are **continuous**
  - Regular DB queries **passive**
    - Client send query – server replies
    - Queries over stream run continuously until they are stopped
  
- **Continuous queries, CQs**

- Active databases (triggers) are related, but:
  - Active database are very side effect oriented (very procedural)
  - The action in a trigger typically updates the database
  - Triggers activated by state changes in a database

- Continuous queries are **non-procedural**
  - CQs can be seen as side effect free functions over streams
  - Database updates are handled as streams injecting tuples into some database (after sufficient data reduction by CQ filters)
Continuous queries

- CQs are turned on and run until stop condition becomes true or when they are explicitly terminated
  - Regular queries executed until result data delivered by demand
- CQs return unbounded data (streams) as result
  - Regular queries bounded by size of tables
- CQs often based on time stamps (logs) of stream elements, i.e. streams contain temporal data
  - Regular queries not temporal
- CQ operators usually montone, i.e. cannot re-read stream, ’just one look’
  - Regular queries can access same table many times to join them
- Non-monotone operators in CQs (e.g. joins) are made over stream windows (i.e. bounded stream segments)
  - Regular DBMS queries specified over entire tables
A simple example

\[ \text{peakdetect} \left( \text{winagg} \left( \text{readstream} \left( \text{"SIG2"} \right), 256, 256 \right) \right); \]
Stream windows

• Limited size section of stream stored temporarily in database
  – A window regarded as a stored regular database table
  – Regular database queries (including joins) can be made over these windows

• Need window operators to chop stream into segments
  – as [RANGE 30 SECONDS] in CQL

• Window size based on:
  – Number of elements, a counting window
    • E.g. last 10 elements
      – i.e. windows has fixed size of 10 elements
  – A time window
    • E.g. elements last second
      – i.e. a window contains all event processed during the last second
  – A landmark window
    • All events from time t₀ in window
      – c.f. log file
Stream windows

- Windows also have **stride**
  - Rule for how fast they move forward,
    - E.g. 10 elements for a 10 element counting window or 10s for a 10s time window
      => A **tumbling** window
    - E.g. 2 elements for a 10 element counting window or 2s for a 10s time window
      => A **sliding** windows

- The CQ language needs **window operators** to collect stream tuples into stream of windows of tuples
  - Stride and window size are parameters
Window joins

- Join over temporary data in window
  - Can be full join with aggregate functions (group-by)
- Window join operators become approximate
  - Since only windows of elements are joined rather than entire stream
    - In a relational database all elements in tables are matched in a join
    - Often approximate summaries of streaming data maintained
      - E.g. moving average, standard deviation, max, min
      - The summaries also become dynamic streams!
Parallel DSMSs

- Classical DSMSs execute on one computer
  - Can be many threads (multi-core)
- The streams contain rather low volume events
  - Rather slow reading from sensors
    - Sensor networks
    - Smart dust
- Typically relatively simple processing (conditions) over events
  - E.g. thresholds
    - Temperature > 100 and pressure < 200
    - CQ typically makes data reduction by source filtering
- Could still be substantial data volumes at the data sink
  - Collect all event into a relational database for SQL analyses
Parallel DSMSs

• New applications for DSMS require
  – Advanced queries and computations on stream elements, e.g.:
    • E.g. applying advanced statistical operators over windows
    • Complex numerical computations (FFT, PCA)
    • Advanced pattern matching
  – Scalability over both data volume and computations

• Solution
  – Parallel DSMS
    • Computations made in parallel
The LOFAR Instrument
- 13000 antennas
- Distributed over 100 stations
- Producing ~20Tbps raw data

UU: Developing a scalable DSMS to process LOFAR stream queries
sa.engine (SCSQ)

• The world’s fastest DSMS 2010-?
• Measured with DSMS benchmark Linear Road: http://www.cs.brandeis.edu/~linearroad/
• Expensive computations on stream at network speed
• Performance obtained by parallelization on clusters
• See: http://user.it.uu.se/~udbl/SCSQ.html
• SVALI: SCSQ extended with features for analyses of streams from industrial equipment
Performance

Linear Road

A Stream Data Management Benchmark

The Linear Road Benchmark makes it possible to compare performance characteristics of Stream Data Management Systems relative to each other and relative to alternative systems like Relational Databases. Stream Data Management Systems process streaming data by executing continuous historical queries while producing query results in real time.

Linear Road has been endorsed as a Stream Data Management Systems Benchmark by the developers of both Aurora (out of Brandeis University, Brown University and Massachusetts Institute of Technology) and STREAM (out of Stanford University) stream systems.

NEW: Linear Road Implementation on PC by Uppsala University

www.cs.brandeis.edu/~linearroad
How: Massively parallel stream processing

AmosQL+

sa.engine
Use case: fleet management and analysis

Distributed equipment
Embedded sa.engine

sa.engine

Communicated data streams

Internet

Analysis cluster

Reducing combining analyzing transforming inferencing

sa.engine

Analyst’s Terminal

Visualization
Search
Reaction

Feedback actions

Data collection/reduction/ transformation
The LOFAR Instrument
-13000 antennas
-Distributed over 100 stations
-Producing ~20Tbps raw data

UU: Developing a scalable DSMS to process LOFAR stream queries
Well known DSMS Systems

STORM (http://storm.incubator.apache.org/): Developed by Twitter. Distributed stream processing engine where programmers specify communicating distributed computations in Java. Java library, no query language.

MillWheel (http://static.googleusercontent.com/media/research.google.com/sv/pubs/archive/41378.pdf): Google’s answer to STORM.

Kinesis (https://aws.amazon.com/kinesis/streams/): Amazon’s answer to STORM.


Overview paper

⇒ L. Golab and T. Özsu: Issues in Stream Data Management, SIGMOD Records, 32(2), June 2003, 
http://www.acm.org/sigmod/record/issues/0306/1.golab-ozsu1.pdf