Data Stream Management vs. Traditional Data Management

- Data is moving! Continuously generated (assumed infinite!)
- At high pace
- Queries are (mainly) continuous (aka. standing). Registered once, observed “forever”.
- Answer to queries in (near) real-time required (often)
- Probabilistic methods for efficiency or considering only part of the stream (sliding window)

DBMS vs. DSMS

<table>
<thead>
<tr>
<th>Database management system (DBMS)</th>
<th>Data stream management system (DSMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent data (relations)</td>
<td>Volatile data streams</td>
</tr>
<tr>
<td>Random access</td>
<td>Sequential access</td>
</tr>
<tr>
<td>One-time queries</td>
<td>Continuous queries</td>
</tr>
<tr>
<td>(theoretically) unlimited secondary storage</td>
<td>Limited main memory</td>
</tr>
<tr>
<td>Only the current state is relevant</td>
<td>Consideration of the order of the input</td>
</tr>
<tr>
<td>Relatively low update rate</td>
<td>Essentially extremely high update rate</td>
</tr>
<tr>
<td>Little or no time requirements</td>
<td>Real-time requirements</td>
</tr>
<tr>
<td>Assumes exact data</td>
<td>Assumes outdated/inaccurate data</td>
</tr>
<tr>
<td>Non-removable query processing</td>
<td>Variable data arrival and data</td>
</tr>
<tr>
<td>characteristics</td>
<td></td>
</tr>
</tbody>
</table>

Overview of Data Stream Topics

- Synopses:
  - concise representations of stream content
  - tailored to tasks, e.g., counting distinct elements
  - usually not exact, but approximations (estimators) of true values.
- Windows:
  - focus of certain recent subset of data
  - computation of functions/joins over window(s) content
  - Think: SQL over stream windows (ranges)
SYNOPSIS (ESTIMATORS)

Counting Occurrences

• Consider a stream of elements \( a_1, a_2, a_3, a_4, a_5, \ldots \)
• How often does \( a_i \) occur?
• How to implement?
• Keep counter for each id
• Required space \#ids = \( N \)
• Not feasible if \( N \) is very large

Probabilistic Counting: Count-Min Sketch

• Keep 2-dim array \((h, r)\)
• \( h \) hash functions* that map to range \( 0...r-1 \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Arriving item \( a \)
• For each \( j \): \( \text{array}[j, h(a)]++ \)


Count-Min Sketch: Counting

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

• How often did we see item \( a \)?
• \( h_1(a) = 4, h_2(a) = 5, h_3(a) = 0, h_4(a) = 2 \)
• Take minimum of the corresponding values in the 2-d array. Here: 4
• Estimate is never underestimating
• Overestimation probabilistically bounded

Unbiased vs. Biased Estimators

• Given a real number \( n \) and an estimator of it, denoted as \( \hat{n} \)
• E.g., number of distinct elements in a set \( S \)
• \( \hat{n} \) is called an unbiased estimator of \( E[\hat{n}] = n \)
• and biased otherwise, in which case \( \text{Bias}[\hat{n}] = E[\hat{n}] - n \)

Counting Distinct Elements

• Consider a stream of elements \( a_1, a_2, a_3, a_4, a_5, \ldots \)
• How to compute/estimate the number of distinct elements observed?
Usability
- Streams (one pass, little memory footprint)
- Distributed systems: compact data exchange (recall Bloom filter)
- Sketches for partial data can be merged for global view

Flajolet Martin (FM) Sketch (aka. Hash Sketch)
- Proposed originally by Flajolet and Martin in 1985
- Allocate a bitvector $B$ of size $m = \log(N)$
- Hash items to bitvector positions according to a geometric distribution:
  - Hash each item $i$ to a m-bit number $h(i)$
  - Compute position $k$ of the least-significant "1" of $h(i)$
  - Set the bit $B[k]$ to "1"
- $S: 17, 5, 19, 211, 17, 5, 31$
- $h(17) = 010100$ then least-sign. 1 bit $= 3$
- $h(5) = 000101$ then least-sign. 1 bit $= 1$

FM-Sketch: Estimator
- In the end $B$ might look like this
  - $B[0]$ is set approximately $n/2$ times
  - $B[1]$ is set approximately $n/4$ times
  - $B[i]$ is set if $i > \log_2(n)$
- "Mix" of 1s and 0s around $i = \log_2(n)$
- Use left-most zero at indicator for $\log_2(n)$:
  - $n \approx 2^{\text{position of left most zero bit}}$

FM Sketch: Intuition/Idea
- $B[0]$ is set approximately $n/2$ times
- $B[1]$ is set approximately $n/4$ times
- $B[i]$ is set if $i > \log_2(n)$
- "Mix" of 1s and 0s around $i = \log_2(n)$
- Use left-most zero at indicator for $\log_2(n)$:
  - $n \approx 2^{\text{position of left most zero bit}}$

FM-Sketch: Union
- Given: two multisets $S$ and $T$ and their sketches $B_S$ and $B_T$ of size $m$
- Then:
  - The sketch $B = B_S \cup B_T$ is the sketch of $S \cup T$

K-Min Value (KMV) Synopsis
- Given set $S$ of values. Want number of distinct elements $n := D(S)$ (notation)
- Hashing outputs values uniformly in $[0,1]$
- KMV synopsis is ordered set of $k$ smallest values
- Unbiased Estimator: $\hat{R}^U_k = (k - 1) / U_k$
  - Exact error analysis based on theory of order statistics
  - Asymptotically optimal as $k$ becomes large
Min-Hashing

- Hash function \( h \) maps elements of a set to integer space. Let’s do that for two sets, \( A \) and \( B \).
- Let \( \min(A) \) and \( \min(B) \) denote the minimum of these numbers for set \( A \) and \( B \), respectively.
- Then
  \[
  P[h_{\min}(A) = h_{\min}(B)] = \frac{|A \cap B|}{|A \cup B|}
  \]

Min-Hashing (Cont’d)

- **Why does it work?** If the min values are the same, that element that causes the min value has to be in \( A \cap B \); probability is \( \frac{|A \cap B|}{|A \cup B|} \).
- Unbiased estimator (but too high variance)
- As seen before for other estimators: improved estimation quality through multiple “rounds” of estimates (Error is \( O(1/k) \), with \( k \) rounds)
  - one min value, multiple hash functions
  - several min values, one hash function

Min-Hash: Estimator

```python
count = 0
for each hash function h:
    if h_{\min}(A) == h_{\min}(B) then
        count++
end

Estimate of Jaccard is given as \( \text{count} / k \)
```

See exercise in assignment sheet 5

Literature

DATA STREAM MANAGEMENT
SYSTEMS AND CQL

Thanks to Johannes Gehrke (Cornell) for providing some of the following material. Many of the slides are initially based on material by Jennifer Widom (Stanford).

Data Stream Model

- A stream $S$ is a (possibly) infinite bag (multiset) of elements $<s,t>$ where $s$ is a tuple belonging to the schema of $S$ and $t$ is the timestamp of the element.
- Think: tuples of a relational DBMS extended with timestamp, streaming in.

Data Streams: Example

- Monitoring of highway traffic:
  $\text{PosSpeedStr}(\text{vehicleId}, \text{speed}, \text{xPos}, \text{dir}, \text{hwy})$
- E.g., for:
  - congestion prediction/warning
  - estimates of travel time
  - toll collection!
  - ticket for too fast driving

Data Streams: Example

- Environmental Monitoring
  $\text{StationStream}(\text{humidity}, \text{solarRadiation}, \text{windSpeed}, \text{snowHeight})$
- Various application scenarios:
  - avalanche risk level computation
  - insights for agriculture
  - air pollution (urban) monitoring

Continuous Queries

- In contrast to ad-hoc, single time queries in (relational) DBMS.
- Queries over Streams are considered continuous: registered once, run "forever":
  - "want to stay updated to avalanche risk, not just check once"
- Also called standing queries or subscriptions (in publish/subscribe context)
- For instance:
  - Compute average temperature.
  - Select all orders of stock "Apple" with quantity larger than 100.

What and How can we Compute DB-Style Queries?

- How to compute average values over an infinite stream? Block forever?
- How to join infinite streams if join partners can arbitrarily arrive (or not)?
- Idea: keep window that renders a continuous (infinite) stream a snapshot/static relation
Sliding Window Concept

- Focus attention to latest values of stream
- Allows computation of aggregates
- Joins are computed across windows overlaid of other (or same) streams

Sliding Window: Example

- Window of size W
  - based on time (=> time-based)
  - or number of tuples inside (count-based)
- Shifted every t by B

Sliding Window Aggregates

- Output average for each window when it slides.
- Here:
  - 17.7°C
  - 26.3°C
  - 19.1°C

Sliding Window Joins

- Join is executed over individual window contents.

Types of Sliding Windows

- Time based Window
  - window contains tuples within a certain time range; e.g., Twitter Tweets of the last 10 minutes, stock market values of the last 10 seconds
  - size can arbitrarily change if input rate changes
- Count-based Window
  - window contains at any time a fixed amount of items, say, the last 100 Tweets or 10000 last stock trades
  - newly arriving items kick out older ones (once window is filled up), depending on strategy (next slide)

Types of Sliding Windows (Cont’d)

- Sliding Window: move window on certain ticks/time, continuous or in blocks
- Tumbling Window: create new window for each time range or size W (i.e., collect data until full or for W time; then reset)
  - At each slide/”tumble” a function can be applied to window content and the result outputted
  - This is also called “trigger”.
Overview of Data Stream Management Systems (DSMSs)

- **STREAM** (Stanford University), **Aurora** (Brandeis/Brown/MIT), **TelegraphCQ** (UC Berkeley), **Cayuga** (Cornell), **PIPES** (Uni Marburg), ...

- Large interest also from companies/startups: Oracle Microsoft, IBM, Streambase

- Lately open-source product for big data distributed streams: Yahoo! S4, Twitter Storm (will see in detail later)

---

**STREAM**

- **Stanford Stream Data Manager**
- “General purpose” DSMS for streams and stored data

- Declarative query language to phrase continuous queries (SQL like).

---

Continuous Query Language – CQL

SQL with:

- Streams
- Windows
- New semantics (stream) – Three relation-to-stream operators: Istream, Dstream, Rstream
- Sampling

---

Example Relation (Used Later)

**Simplified Linear Road Setup:**

- A single input stream: The stream of positions and speeds of vehicles

  PosSpeedStr(vehicleId, speed, xPos, dir, hwy)

- vehicleId: vehicle
- speed: speed in MPH
- xPos: Position of the vehicle within the highway in feet
- dir: direction (east or west)
- hwy: highway number

---

Example Query 1

- Two streams:
  - Orders (orderID, customer, cost)
  - Fulfillments (orderID, clerk)

- Total cost of orders fulfilled over the last day by clerk “Sue” for customer “Joe”

  ```sql
  SELECT sum(O.cost)
  FROM Orders O, Fulfillments F [Range 1 Day]
  WHERE O.orderID = F.orderID and F.clerk = "Sue" and O.customer = "Joe"
  ```
Example Query 2
• Using a 10% sample of the fulfillments stream, take the 5 most recent fulfillments for each clerk and return the maximum cost

```
SELECT F.clerk, max(O.cost)
FROM orders O,
    fulfillments F
    [PARTITION BY clerk ROW 5] 10% SAMPLE
WHERE O.orderID = F.orderID
GROUP BY F.clerk
```

CQL: Relations and Streams
• T: discrete, ordered time domain
• A relation R is a mapping from time T to bag of tuples belonging to the schema of R.
• That is, R(t) varies over time
• Updates carry timestamps, too!
• A stream is a set of \( (\text{tuple}, \text{timestamp}) \) elements

Streams \( \rightarrow \) Relations

Streams

Window specification

Relations

Special operators:
Istream, Dstream, Rstream

Any relational query language

Stream \( \rightarrow \) Relation

• \( S [W] \) is a relation - at time \( T \) it contains all tuples in window \( W \) applied to stream \( S \), up to time \( T \).
• When \( W = \infty \), it contains all tuples in stream \( S \) up to time \( T \)
• Ways to construct these windows \( \{W\} \)
  – Time-based
  – Tuple-based
  – Partitioned

Time-Based Window

• \( S \) [Range \( T \)]
  – \( S \) [Now]
  – \( S \) [Range Unbounded]

Examples:
• PosSpeedStr [RANGE 30 Seconds]
• PosSpeedStr [NOW]
• PosSpeedStr [RANGE Unbounded]

Tuple-Based Window

• \( S \) [Rows \( N \)]
  – If tuples form a partial order, ties are broken arbitrarily
  – [Rows Unbounded]

Example:
• PosSpeedStr [ROWS 1]
Partitioned Windows

- \( S \) [Partition By \( A_1, \ldots, A_k \) Rows N]
  1. Logically partition \( S \) into substreams (compare to SQL GROUP By)
  2. Compute a tuple sliding window
  3. Take union

Example:
- Recall: PosSpeedStr
  - \( \text{vehicleId} \), \( \text{speed} \), \( \text{xPos} \), \( \text{dir} \), \( \text{hwy} \)
- \( \text{PosSpeedStr} \) [PARTITION BY \( \text{vehicleId} \) ROWS 1]

Relation \( \rightarrow \) Relation

- With previous window transform we get a relation, now we can apply any query expressed in SQL — just that deal now with time-varying relations

Example:
- \( \text{SELECT distinct vehicleId} \)
  - FROM PosSpeedStr \( \text{[RANGE} \) 30 Seconds] \( \text{]} \)

Computes the active vehicles

Relation \( \rightarrow \) Stream

- \( \text{Istream}(R) \) contains a stream element \((r,t)\)
  whenever \( r \) in \( R(t) \) \( \setminus \) \( R(t-1) \) \( \text{“Insert stream”} \)
- \( \text{Dstream}(R) \) contains a stream element \((r,t)\)
  whenever \( r \) in \( R(t-1) \) \( \setminus \) \( R(t) \) \( \text{“Delete stream”} \)
- \( \text{Rstream}(R) \) contains a stream element \((r,t)\)
  whenever \( r \) in \( R(t) \) \( \text{“Relation stream”} \)

Istream, Dstream, and Rstream

- \( \text{Istream}(R) \): contains all tuples in \( R \) that are new within the last time period, i.e., insert stream
- \( \text{Dstream}(R) \): contains all tuples in \( R \) which where in the stream before the last period (and not anymore in now), i.e., delete stream
- \( \text{Rstream}(R) \): contains all tuples in \( R \)

Note: \( \text{Istream} \) and \( \text{Dstream} \) are expressible with \( \text{Rstream} \) and suitable selections. How?

Relation \( \rightarrow \) Stream: Examples

- \( \text{SELECT Istream(*)} \)
  - FROM PosSpeedStr \( \text{[RANGE Unbounded]} \)
  - WHERE speed > 65

- \( \text{SELECT Rstream(*)} \)
  - FROM PosSpeedStr \( \text{[NOW]} \)
  - WHERE speed > 65

Query Results at Time T

- Use all relations at time \( T \)
- Use all streams up to \( T \), converted to relations
- Compute relational results
- Convert result to streams if desired
Examples

• What is the following query doing?

```sql
SELECT Istream(Avg(A)) FROM S [Range 5 seconds]
```

Intention: Emit 5-second moving average on every timestep, but output is generated only if average changes (Istream!)

• To emit a result on every timestep

```sql
SELECT Rstream(Avg(A)) FROM S [Range 5 seconds]
```

• To emit a result on every second

```sql
SELECT Rstream(Avg(A)) FROM S [Range 5 seconds Slide 1 second]
```

Examples (Cont’d)

SELECT F.clerk, max(O.cost) FROM O [∞], F [Rows 1000] WHERE O.orderID = F.orderID GROUP BY F.clerk

• At time T: entire stream O and last 1000 tuples of F as relations
• Evaluate query, update result relation at T

Query Execution in STREAM

• When a continuous query is registered, generate a query execution plan
  — New plan merged with existing plans
  — Users can also create & manipulate plans directly
• Plans composed of three main components:
  — Operators
  — Queues (input and inter-operator)
  — State (windows, operators requiring history)
• Global scheduler for plan execution

Simple Query Plan

More Topics

• Seen only formal model and standard concepts of data stream management systems
• There is of course much more to it
• Implementation, optimization (e.g., equivalences), load shedding, ...
• Will be an own lecture by itself.
• Next, look at system aspects in distributed data stream management systems and (mobile) sensor networks
Literature

- Jürgen Kramer, Bernhard Seeger: Semantics and implementation of continuous sliding window queries over data streams. ACM Trans. Database Syst. 34(1) (2009)