Distributed Data Management
Summer Semester 2013
TU Kaiserslautern

Dr.-Ing. Sebastian Michel

smichel@mmci.uni-saarland.de
Lecture 8+

(DISTRIBUTED) DATA STREAM PROCESSING (INTRODUCTION)
So Far: Databases/NoSQL Datastores

• Data is changing, yes, but this is more due to inserts and update to stored data items
• Historic data is kept
• Queries operate on full data (tables)
• MapReduce is extreme, Write-once & Read-many times
• Data warehousing, too: periodically loading data in store for deep(er) analytics
• Data mining
Data Stream Management vs. Traditional Data Management

- At query time, data is accessed as a whole
- Data is persistently stored
- Queries are ad-hoc (mainly)

Distributed Data Management, SoSe 2013, S. Michel
Data Stream Management vs. Traditional Data Management (Cont’d)

- 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)
Sensor Networks

- (Distributed) Sensor Networks; “Smart Dust”
- Mainly numeric measurements of (natural) phenomena
- Computation of queries like max, average, min, quantiles, value > τ
Mobile Ad-Hoc Networks

• Connection between sensors are ad-hoc
• Efficient and reliable routing
• Understanding (changing) topology
• Power consumption
• Also: Vehicular Ad-Hoc Netw.
Social Sensors

• Explicitly: Snow Tweets (http://snowcore.uwaterloo.ca/snowtweets/)
  – #snowtweets 50.0 cm. at K1A 0A2
  – #snowtweets 10.0 in. at 20500
  – #snowtweets 4 cm at Palmerston North 4414

• Implicitly: By mentioning topics, people, in social communication
Earthquake News on Twitter

source: http://blog.socialflow.com/
Stock Market

• Real-time analysis of stock marked changes
• Computing statistics over streams, e.g., for decision support
• Opportunities for reacting in real-time
• Even with fully automated means: algorithmic trading
## 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>potentially 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>Plannable query processing</td>
<td>Variable data arrival and data</td>
</tr>
<tr>
<td></td>
<td>characteristics</td>
</tr>
</tbody>
</table>

Data Stream Model

• Stream of data items is unbounded (available memory is not)

• No way to store entire stream (how could we, its (probably) not ending)

• To compute query results, need to devise algorithm with little memory consumption
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
Data Stream Mining: Teasers

• I tell you integer numbers between 1 and N
• I will tell all but one number

  481  324  122  412  871  231  849  447  641 ...

• After N-1 numbers I ask: which number was missing.
Data Stream Mining: Teasers (Cont’d)

• Keep Boolean array of length N:
  – Mark position for observed number
  – Size required: N
  – Computation at end: N to find missing number

• Much better:
  – keep sum of numbers: S
  – Missing number is N*(N+1)/2 - S
Counting Occurrences

- Consider a stream of elements \( a_i \)
  
  \[ \ldots, a_2, a_{84}, a_{41}, a_2, a_{77}, a_{231}, a_2, a_4, a_{54}, \ldots \]

- How often does \( a_2 \) 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)

0 1 2 3 4 5

\[ \begin{array}{ccccccc}
\ h_1 \\
\ h_2 \\
\ h_3 \\
\ h_4 \\
\end{array} \]

• Arriving item \( a \)

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

### 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>$h_1$</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>$h_3$</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$h_4$</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
Outlook

• Some more basics of stream processing
• Few more fundamentals of data stream mining
• Then, window based data stream management
• Then going distributed:
  – Distributed DSMS
  – Distributed (ad-hoc) sensor networks
  – Scalable computation of massive data streams (think: Hadoop for data streams)