Data Stream Management

Two driving forces:

- A collection of applications where data streams naturally exist but DBMS doesn’t help much
- Advances of microsensor technologies
Data Streams

- **Financial services**
  - Data feeds: stock tickers, foreign exchange transactions…
  - Data rate: thousands of messages per second
  - Applications: routing trade requests, automating trade strategies, market trend analysis…
  - **StreamBase:**
Data Streams (cont’d)

- Network monitoring
  - Packet traces, network performance measurements…
  - Data rate: several megabits to gigabits per second
  - Applications: traffic analysis (e.g. fraction of traffic on an ISP link coming form a customer network), performance monitoring, router configuration, intrusion detection…

- System/Application monitoring…
Wireless Sensor Networks

- **Wireless sensor networks**
  - Sensor devices: sense temperature, pressure, acceleration, humidity, magnetic field, ...
  - A set of sensor devices auto-configure themselves into a communication network
  - Applications:
    - environment monitoring
    - habitat monitoring
    - structural monitoring
    - vehicle tracking...
  - TinyDB: [http://telegraph.cs.berkeley.edu/tinydb/](http://telegraph.cs.berkeley.edu/tinydb/)

Yanlei Diao, University of Massachusetts Amherst 4/26/2006
Radio Frequency Identification

- **RFID Technologies**
  - Tags: small devices, transmitting a tag id when brought close to a reader
  - Readers: interrogate tags with radio signal; read constantly, over a range, without line-of-sight
  - Applications:
    - supply chain management
    - healthcare
    - public transportation
    - postal services
    - pharmaceuticals…
  - EPCGlobal: [http://www.epcglobalinc.org/](http://www.epcglobalinc.org/)

---

Yanlei Diao, University of Massachusetts Amherst
Why Not Use DBMS?

- *Human-Active, DBMS-Passive*: passive data repository, queries initiated by humans
- Data is mostly about current state
- Triggers and alerters are second-class citizens
- Assuming that data is up-to-date and query have exact answers
- No real-time services

[Carney et al., VLDB’02]
What is Needed for Streams?

- **DBMS-Active, Human-Passive**: data comes from external sources not humans, DBMS detects activity of interest and alerts humans
- Querying over history, time-series analysis
- Monitoring applications are trigger-oriented; scale of trigger processing is large
- Data is incomplete and stale; query processing returns approximate answers
- Real-time requirements, Quality-of-Service
Execution Models: A Comparison

Traditional DBMS

- Data persistently stored
- One-time or periodic queries
- Query-driven execution

Data Stream Processor

- Data in motion, unending
- Continuous, long-running queries
- Data-driven execution

<table>
<thead>
<tr>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Yanlei Diao, University of Massachusetts Amherst
Main Issues of Stream Management

- Query languages
- Single query processing
- Multi-query optimization
- Adaptive query processing
- Real-time processing
- Approximate answers
- ...
Query Languages

- Extensions of SQL: SEQUIN [SLR’96], CQL [ABW03], TCQ [CCD+03], GSQL [CJS+03]
- Continuous Query Language (CQL)
  - FROM clause can address both streams and relations
  - PARTITION BY clause creates sub-streams
  - Sliding windows: ROW, physical window by number of tuples; RANGE, logical window in terms of seconds, minutes, or hours; SLIDE, window movement

**Sliding window join:**
Select *  
From S1 [Rows 5], S2 [Rows 10]  
Where S1.a = S2.a

**Streaming aggregates:**
Select Count(*)  
From S [Range 1 Minute]  
Select Count(*)  
From S [Range 1 Minute Slide 1 Minute]
Single Query Processing

- **Blocking** query operator
  
  "A query operator that is unable to produce the first tuple of its output until it has seen its entire input." [BBD+02]
  
  - Sorting?
  - Sort-merge join?
  - Aggregate operators, min(), max(), count()?
  - Nested-loops join?
  - Hash join?

- **Non blocking** query operator
  
  "The operator does not stage data (either in memory or on disk) without producing results for a long time." [UF01]
  
  - produces the initial result early
  - returns results tuples incrementally as they become available
Blocking Join Operator

• Traditional Hash Joins block when one input stalls.
Non-Blocking Join Operator

- Symmetric Hash Join (SHJ) blocks only if both stall.
- Processes tuples as they arrive from sources.
- Produces all tuples in the join and no duplicates.
Non Blocking Join with Limited Memory

- SHJ requires both inputs to be memory resident.
  - High memory overhead if inputs are large.
  - For a complex query, this means all intermediate results must be in memory.

- XJoin extends SHJ to allow it to work with limited memory [UF00].
  - Partition each input using a hash function.
  - When allocated memory is exhausted, a partition is flushed to disk. So at each point, a partition = disk-resident data (older) + memory-resident data (newer)
  - Join processing continues on memory-resident data.
  - Disk-resident tuples are handled in background.
XJoin: Handling the Partitions

Memory-resident partitions of source A

\[ \text{hash(Tuple A)} = 1 \]

Memory-resident partitions of source B

\[ \text{hash(Tuple B)} = n \]

Disk-resident partitions of source A

Disk-resident partitions of source B

SOURCE-A

SOURCE-B

Yanlei Diao, University of Massachusetts Amherst

4/26/2006
Stage 1: Memory-to-Memory Joins

Partitions of source A

\[ \text{hash(record A)} = i \]

SOURCE-A

Partitions of source B

\[ \text{hash(record B)} = j \]

SOURCE-B

Tuple A

build

probe

Output

Tuple B
Stage 2: Disk-to-Memory Joins

Partitions of source A

Partitions of source B

Output

Partitions of source A

Partitions of source B

Yanlei Diao, University of Massachusetts Amherst

4/26/2006
Three Stages of XJoin

- **Stage 1 - Symmetric hash join (memory-to-memory) with partitions**
- **Stage 2 - Disk-to-memory**
  - Separate thread - runs when stage 1 blocks (both input stall).
  - Stage 1 and 2 interleave until all input has been received.
- **Stage 3 - Clean up stage**
  - Stage 1 misses pairs that were not in memory concurrently.
  - Stage 2 misses pairs when both are on disk, and may not get to run to completion.
  - Stage 3 joins all the partitions (memory-resident and disk-resident portions) of the two sources.
Windowing Operators

- **Windowed joins**
  - Based on non-blocking symmetric hash join
  - Adding support for windows
    - Window size: number of tuples, logical time
    - Sliding of window: smooth movement or hopping
    - Maintain right state (tuples residing in the window) for join
  - [CF02, MAF+03, GO03]

- **Windowed aggregates**
  - Windows are necessary for computing aggregates
  - Ideas borrowed from sequence databases
  - [SLR96, GKS01, DGI+02]
Multi-Query Optimization

- In streaming environments, many standing queries naturally exist.
  - Queries can be as simple as a predicate or highly complex involving many joins.
  - The number of queries can reach thousands or even more, e.g., one query per user of a financial service.
- For **efficiency** and **scalability**, handle all queries together sharing the work when possible.
  - Multi-query optimization (MQO) is a NP-hard problem.
  - Most work on simple selections and joins [HCH+99, CDT+00, CDN02, MSH+02].
  - Recent work on sharing windowed joins [HFA+03] and windowed aggregates [KWF06].
NiagaraCQ [CDT+00]

- A continuous query system for the Internet
- Scalable CQ processing: supporting large numbers of queries
- Exploiting sharing of queries
  - Simple selections
  - Simple joins
- Key ideas:
  - Grouping queries based on signatures
  - Constructing group query plans, i.e., a shared plan per group
  - Supporting incremental group optimization upon query updates
Query Signatures

- Use of expression signatures for grouping
  - Same syntax structure
  - Different constant values

- Query examples
  Q1: `doc("quotes.xml")/Quotes/Quote [Symbol = "INTC"]`
  Q2: `doc("quotes.xml")/Quotes/Quote [Symbol = "MSFT"]`

- Expression signature

```
Quotes/Quote/Symbol
```

in quotes.xml

constant
Individual Query Plans

Trigger Action I
- Select Symbol = “INTC”
- File Scan
  - quotes.xml

Trigger Action J
- Select Symbol = “MSFT”
- File Scan
  - quotes.xml
**A Query Group**

- **Group Signature**
  - Common signature of all queries in the group

- **Group constant table**

<table>
<thead>
<tr>
<th>Constant_value</th>
<th>Dest_buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>INTC</td>
<td>Dest. I</td>
</tr>
<tr>
<td>MSFT</td>
<td>Dest. J</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Group Plan

- Trigger Action I
- Split
  - Join
    - Symbol = constant
    - File Scan:
      - quotes.xml
    - File:
      - Constant table
- Trigger Action J

Constant | Intermediate file name
---|---
.... | ....
INTC | file. i
MSFT | file. j
.... | ....
Queries with Joins

- Example queries
  Q3: for $q$ in doc("quotes.xml")/Quotes/Quote[Change_Ratio>0.2],
     $c$ in doc("companies.xml")/Companies/Company[Symbol = $q/Symbol]
     return ($q, $c)

  Q4: for $q$ in doc("quotes.xml")/Quotes/Quote[Change_Ratio>0.5],
     $c$ in doc("companies.xml")/Companies/Company[Symbol = $q/Symbol]
     return ($q, $c)

- Expression Signature

\[
\begin{align*}
\text{Quotes/Quote/Change\_Ratio} & = \text{constant} \\
\text{Companies/Company/Symbol} & = \text{../Symbol in companies.xml}
\end{align*}
\]
**Individual Query Plans**

- Trigger Action I
  - Join on Symbol
    - Select Symbol = “INTC”
      - File Scan quotes.xml
    - File Scan companies.xml
- Trigger Action I
  - Join on Symbol
    - Select Symbol = “MSFT”
      - File Scan quotes.xml
    - File Scan companies.xml
Group Plan

Trigger Action I

Split

Join
Symbol = constant

Join on Symbol

File Scan
quotes.xml

File Scan
companies.xml

File
Constant table

Trigger Action J

Selection pull-up to allow sharing of the join!
Incremental Grouping

When a new query is submitted:

If the expression signature of the new query matches that of existing groups

Break the query plan into two parts
Remove the lower common part
Add the upper part onto the group plan

else create a new group
Other Research

- Query languages
  - SEQUIN [SLR96], CQL [ABW03], TCQ [CCD+03], GSQL [CJS+03]

- Single query processing
  - SHJ [HS93], XJoin [UF00], Windowed joins [CF02, MAF+03, GO03], Windows aggregates [SLR96, GKS01, DGI+02]

- Multi-query optimization
  - Simple selections and joins [HCH+99, CDT+00, CDN02, MSH+02], windowed joins [HFA+03] and aggregates [KWF06].

- Adaptive query processing
  - Tukwila [IFF+99], TCQ [CCD+03], STREAM [MWA+03]

- Real-time processing
  - Controlled quality-of-service: Aurora [CBB+03]

- Approximate answers
  - STREAM [MWA+03], Gigascope [JMR04, CJK+04], [GKS01, DGG+02]

...