Querying Sensor Networks

Sam Madden
UC Berkeley
Mangled by Andreas Terzis
Overview

- TinyDB: Queries for Sensor Nets
- Processing Aggregate Queries (TAG)
- Taxonomy & Experiments
- Acquisitional Query Processing
Declarative Queries

• Writing sensor network applications challenging
  - Especially for non-CS people
• Users specify the data they want
  - Simple, SQL-like queries
  - Using predicates, not specific addresses
• Challenge is to provide:
  - Expressive & easy-to-use interface
  - High-level operators
    » Well-defined interactions
    » “Transparent Optimizations” that many programmers would miss
      • Sensor-net specific techniques
  - Power efficient execution framework
TinyDB Architecture

SELECT AVG(temp) WHERE light > 400

Queries

Aggavg(temp)

Results

T:1, AVG: 225
T:2, AVG: 250

Multihop Network

Query Processor

Schema:
• “Catalog” of commands & attributes

~10,000 Lines Embedded C Code
~5,000 Lines (PC-Side) Java
~3200 Bytes RAM (w/ 768 byte heap)
~58 kB compiled code
(3x larger than 2nd largest TinyOS Program)
Declarative Queries for Sensor Networks

“Find the sensors in bright nests.”

Examples:

SELECT nodeid, nestNo, light
FROM sensors
WHERE light > 400
EPOCH DURATION 1s

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Nodeid</th>
<th>nestNo</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>17</td>
<td>455</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>25</td>
<td>389</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>17</td>
<td>422</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>25</td>
<td>405</td>
</tr>
</tbody>
</table>
### Aggregation Queries

**2.** SELECT AVG(sound) 
FROM sensors 
EPOCH DURATION 10s

**3.** SELECT region, CNT(occupied), AVG(sound) 
FROM sensors 
GROUP BY region 
HAVING AVG(sound) > 200 
EPOCH DURATION 10s

```
<table>
<thead>
<tr>
<th>Epoch</th>
<th>region</th>
<th>CNT(...)</th>
<th>AVG(...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>North</td>
<td>3</td>
<td>360</td>
</tr>
<tr>
<td>0</td>
<td>South</td>
<td>3</td>
<td>520</td>
</tr>
<tr>
<td>1</td>
<td>North</td>
<td>3</td>
<td>370</td>
</tr>
<tr>
<td>1</td>
<td>South</td>
<td>3</td>
<td>520</td>
</tr>
</tbody>
</table>
```

Regions w/ AVG(sound) > 200

"Count the number occupied nests in each loud region of the island."
Overview

- TinyDB: Queries for Sensor Nets
- Processing Aggregate Queries (TAG)
- Taxonomy & Experiments
- Acquisitional Query Processing
Tiny Aggregation (TAG)

- In-network processing of aggregates
  - Common data analysis operation
    » Aka *gather* operation or *reduction* in || programming
  - Communication reducing
    » Operator dependent benefit
  - Across nodes during same epoch

- Exploit query semantics to improve efficiency!

Madden, Franklin, Hellerstein, Hong. Tiny A Ggregation (TAG), OSDI 2002.
Query Propagation Via Tree-Based Routing

- Tree-based routing
  - Used in:
    » Query delivery
    » Data collection
  - Topology selection is important; e.g.
    » LEACH/SPIN, Heinzelman et al. MOBICOM 99
    » SIGMOD 2003
  - Continuous process
    » Mitigates failures
Basic Aggregation

- **In each epoch:**
  - Each node samples local sensors once
  - Generates *partial state record (PSR)*
    - local readings
    - readings from children
  - Outputs PSR during assigned *comm. interval*

- At end of epoch, PSR for whole network output at root
- New result on each successive epoch

- **Extras:**
  - Predicate-based partitioning via GROUP BY
**Illustration: Aggregation**

**SELECT COUNT(*) FROM sensors**

```
<table>
<thead>
<tr>
<th>Interval #</th>
<th>Sensor #</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```

**Interval 4**

**Epoch**
Illustration: Aggregation

SELECT COUNT(*)
FROM sensors

<table>
<thead>
<tr>
<th>Sensor #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval #</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Interval 3
Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```

Sensor #

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

Interval 2
Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```

Sensor #

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

Interval 1
Illustration: Aggregation

```
SELECT COUNT(*)
FROM sensors
```

Sensor #

Interval #

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

Interval 4

```
1 -> 2
1 -> 3
1 -> 4
1 -> 5
```
Interval Assignment: An Approach

SELECT COUNT(*)

4 intervals / epoch
Interval # = Level

Level = 1

- CSMA for collision avoidance
- Time intervals for power conservation
- Many variations (e.g. Yao & Gehrke, CIDR 2003)
- Time Sync (e.g. Elson & Estrin OSDI 2002)
Overview

- TinyDB: Queries for Sensor Nets
- Processing Aggregate Queries (TAG)
- Taxonomy & Experiments
- Acquisitional Query Processing
Simulation Environment

• Evaluated TAG via simulation

• **Coarse grained event based simulator**
  - Sensors arranged on a grid
  - Two communication models
    » Lossless: All neighbors hear all messages
    » Lossy: Messages lost with probability that increases with distance

• **Communication (message counts) as performance metric**
Taxonomy of Aggregates

• TAG insight: classify aggregates according to various functional properties
  - Yields a general set of optimizations that can automatically be applied

<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial State</td>
</tr>
<tr>
<td>Monotonicity</td>
</tr>
<tr>
<td>Exemplary vs. Summary</td>
</tr>
<tr>
<td>Duplicate Sensitivity</td>
</tr>
</tbody>
</table>

Drives an API!
Partial State

- Growth of PSR vs. number of aggregated values (n)
  - Algebraic: $|\text{PSR}| = 1$ (e.g. MIN)
  - Distributive: $|\text{PSR}| = c$ (e.g. AVG)
  - Holistic: $|\text{PSR}| = n$ (e.g. MEDIAN)
  - Unique: $|\text{PSR}| = d$ (e.g. COUNT DISTINCT)
    - $d = \# \text{ of distinct values}$
  - Content Sensitive: $|\text{PSR}| < n$ (e.g. HISTOGRAM)

“Data Cube”, Gray et. al

<table>
<thead>
<tr>
<th>Property</th>
<th>Examples</th>
<th>Affects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial State</td>
<td>MEDIAN : unbounded, MAX : 1 record</td>
<td>Effectiveness of TAG</td>
</tr>
</tbody>
</table>
Benefit of In-Network Processing

Simulation Results
2500 Nodes
50x50 Grid
Depth = ~10
Neighbors = ~20
Uniform Dist.

![Graph showing Total Bytes Xmitted vs. Aggregation Function]

- **Aggregate & depth dependent benefit!**

- **External**
- **Max**
- **Average**
- **Distinct**
- **Median**

- **Holistic**
- **Unique**
- **Distributive**
- **Algebraic**
## Monotonicity & Exemplary vs. Summary

<table>
<thead>
<tr>
<th>Property</th>
<th>Examples</th>
<th>Affects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial State</td>
<td>MEDIAN: unbounded, MAX: 1 record</td>
<td>Effectiveness of TAG</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>COUNT: monotonic, AVG: non-monotonic</td>
<td>Hypothesis Testing, Snooping</td>
</tr>
<tr>
<td>Exemplary vs. Summary</td>
<td>MAX: exemplary, COUNT: summary</td>
<td>Applicability of Sampling, Effect of Loss</td>
</tr>
</tbody>
</table>
Channel Sharing ("Snooping")

- Insight: Shared channel can reduce communication
- Suppress messages that won’t affect aggregate
  - E.g., MAX
  - Applies to all exemplary, monotonic aggregates
- Only snoop in listen/transmit slots
  - Future work: explore snooping/listening tradeoffs
Hypothesis Testing

• **Insight**: Guess from root can be used for suppression
  - E.g. 'MIN < 50'
  - Works for **monotonic & exemplary** aggregates
    » Also **summary**, if imprecision allowed

• **How is hypothesis computed?**
  - Blind or statistically informed guess
  - Observation over network subset
Experiment: Snooping vs. Hypothesis Testing

- Uniform Value Distribution
- Dense Packing
- Ideal Communication

Messages/Epoch vs. Network Diameter
(SELECT MAX(attr), R(attr) = [0, 100])

- No Guess
- Guess = 50
- Guess = 90
- Snooping

Pruning at Leaves
Pruning in Network
Use Multiple Parents

- Use graph structure
  - Increase delivery probability with no communication overhead
- For *duplicate insensitive* aggregates, or
- Aggs expressible as sum of parts
  - Send (part of) aggregate to all parents
    » In just one message, via multicast
  - Assuming independence, decreases variance

\[
P(\text{link xmit successful}) = p \\
P(\text{success from } A \to R) = p^2 \\
E(\text{cnt}) = c \cdot p^2 \\
\text{Var}(\text{cnt}) = c^2 \cdot p^2 \cdot (1 - p^2) = \mathcal{V}
\]

\[
# \text{of parents} = n \\
E(\text{cnt}) = n \cdot (c/n \cdot p^2) \\
\text{Var}(\text{cnt}) = n \cdot (c/n)^2 \cdot p^2 \cdot (1 - p^2) = \mathcal{V}/n
\]

\[n = 2\]
Multiple Parents Results

• Better than previous expectation!
• Losses aren't independent!
• Insight: spreads data over many links

Benefit of Result Splitting

<table>
<thead>
<tr>
<th>COUNT query</th>
<th>Avg. COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splitting</td>
<td></td>
</tr>
<tr>
<td>No Splitting</td>
<td></td>
</tr>
</tbody>
</table>

Critical Link!

No Splitting

With Splitting

6 parents per
Overview

- TinyDB: Queries for Sensor Nets
- Processing Aggregate Queries (TAG)
- Taxonomy & Experiments
- Acquisitional Query Processing
Acquisitional Query Processing (ACQP)

- Closed world assumption does not hold
  - Could generate an infinite number of samples
- An acquisitional query processor controls
  - when,
  - where,
  - and with what frequency data is collected!
- Versus traditional systems where data is provided a priori

ACQP: What’s Different?

- How should the query be processed?
  - Sampling as a first class operation
- How does the user control acquisition?
  - Rates or lifetimes
  - Event-based triggers
- Which nodes have relevant data?
  - Index-like data structures
- Which samples should be transmitted?
  - Prioritization, summary, and rate control
Operator Ordering: Interleave Sampling + Selection

SELECT light, mag
FROM sensors
WHERE pred1(mag)
AND pred2(light)
EPOCH DURATION 1s

At 1 sample / sec, total power savings could be as much as 3.5mW →
Comparable to processor!

Traditional DBMS

\( \sigma(\text{pred1}) \)
\( \sigma(\text{pred2}) \)

\( \sigma(\text{pred1}) \)
\( \sigma(\text{pred2}) \)

\( \sigma(\text{pred1}) \)
\( \sigma(\text{pred2}) \)

Temporary DBMS

ACQP

Correct ordering (unless pred1 is very selective and pred2 is not):

Cheap

Costly

mag
light

mag
light

mag
light
Exemplary Aggregate Pushdown

```
SELECT WINMAX(light, 8s, 8s)
FROM sensors
WHERE mag > x
EPOCH DURATION 1s
```

- Novel, general pushdown technique
- Mag sampling is the most expensive operation!
Lifetime Queries

• Lifetime vs. sample rate
  SELECT ...
  EPOCH DURATION 10 s

  SELECT ...
  LIFETIME 30 days

• Extra: Allow a MAX SAMPLE PERIOD
  - Discard some samples
  - Sampling cheaper than transmitting
Voltage vs. Time, Measured Vs. Expected

Lifetime Goal = 24 Weeks (4032 Hours, 15 s / sample)

Voltage (Raw Units)

Voltage (Expected)

Voltage (Measured)

Linear Fit

R² = 0.8455

Insufficient Voltage to Operate (V = 350)