Acquisitional Query Processing in TinyDB

Sam Madden
UC Berkeley

NEST Winter Retreat 2003
Acquisitional Query Processing (ACQP)

• Cynical DB person question: what’s really different about sensor networks?
  - Low Power? Laptops!
  - Lots of Nodes? Distributed DBs!
  - Limited Processing Capabilities? Moore’s Law!

Being a little bit facetious, but...
Answer

• Long running queries on physically embedded devices that control when and with what frequency data is collected!

• Versus traditional systems where data is provided a priori

Data collection aware query processing “acqusitional query processing”, or ACQP!
ACQP: What’s Different?

• How does the user control acquisition?
  - Rates or lifetimes
  - Event-based triggers

• How should the query be processed?
  - Sampling as a first class operation
  - Events or joins

• Which nodes have relevant data?
  - Semantic Routing Tree
    » Nodes that are queried together route together

• Which samples should be transmitted?
  - Pick most “valuable”? Aggregate others?
  - Store results for later delivery?

SIGMOD Submission!
Outline

• TinyDB
• Acquisitional Language Features
  – Events
  – Buffers
  – Rates & Lifetimes
• Acquisitional Processing (a taste)
  – Optimization of selections
  – Buffering results
    » Choosing where to place storage
Outline

- **TinyDB**
- **Acquisitional Language Features**
  - Events
  - Buffers
  - Rates & Lifetimes
- **Acquisitional Processing (a taste)**
  - Optimization of selections
  - Buffering results
    » Choosing where to place storage
TinyDB/GSK

• Programming sensor nets is hard!
• Declarative queries are easy
  – **TinyDB**: In-network processing via declarative queries
• Example:
  » Vehicle tracking application
    • Custom code
      - 1-2 weeks to develop
      - Hundreds of lines of C
    • TinyDB query (on right):
      - 2 minutes to develop
      - Comparable functionality

```sql
SELECT nodeid
FROM sensors
WHERE mag > thresh
```

EPOCH DURATION 64ms
TinyDB Features

• A distributed query processor for networks of Mica motes
  - Available today!
• Goal: Eliminate the need to write C code for most TinyOS users
• Features
  - Declarative queries
  - Temporal + spatial operations
  - Multihop routing
  - In-network storage
TinyDB Execution

(Aprkost) All Queries are Continuous and Periodic
Written in SQL-Like Language With Extensions For:
- Sample rate
- Offline delivery
- Temporal Aggregation
Declarative Queries for Sensor Networks

• **Examples:**

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

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Nodeid</th>
<th>Light</th>
<th>Temp</th>
<th>Accel</th>
<th>Sound</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>455</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>389</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>422</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>405</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>
Aggregation Queries

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

3. SELECT roomNo, AVG(sound) FROM sensors
   GROUP BY roomNo
   HAVING AVG(sound) > 200
   EPOCH DURATION 10s

Rooms w/ sound > 200

<table>
<thead>
<tr>
<th>Epoch</th>
<th>AVG(sound)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>440</td>
</tr>
<tr>
<td>1</td>
<td>445</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Epoch</th>
<th>roomNo</th>
<th>AVG(sound)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>360</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>520</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>370</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>520</td>
</tr>
</tbody>
</table>
TinyDB Screenshot

```
SELECT s.nodeid, s.light FROM sensors AS s
EPOCH DURATION 1024
```

<table>
<thead>
<tr>
<th>Epoch</th>
<th>nodeid</th>
<th>light</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>12</td>
<td>836</td>
</tr>
<tr>
<td>26</td>
<td>16</td>
<td>859</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>846</td>
</tr>
<tr>
<td>26</td>
<td>8</td>
<td>923</td>
</tr>
<tr>
<td>26</td>
<td>13</td>
<td>523</td>
</tr>
<tr>
<td>27</td>
<td>16</td>
<td>851</td>
</tr>
<tr>
<td>27</td>
<td>8</td>
<td>915</td>
</tr>
<tr>
<td>27</td>
<td>13</td>
<td>541</td>
</tr>
<tr>
<td>27</td>
<td>12</td>
<td>844</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>838</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>838</td>
</tr>
<tr>
<td>28</td>
<td>12</td>
<td>836</td>
</tr>
<tr>
<td>28</td>
<td>13</td>
<td>528</td>
</tr>
<tr>
<td>28</td>
<td>16</td>
<td>852</td>
</tr>
<tr>
<td>28</td>
<td>8</td>
<td>923</td>
</tr>
<tr>
<td>29</td>
<td>13</td>
<td>536</td>
</tr>
<tr>
<td>29</td>
<td>12</td>
<td>843</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>837</td>
</tr>
<tr>
<td>29</td>
<td>16</td>
<td>851</td>
</tr>
<tr>
<td>30</td>
<td>13</td>
<td>523</td>
</tr>
<tr>
<td>30</td>
<td>16</td>
<td>859</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>846</td>
</tr>
<tr>
<td>31</td>
<td>8</td>
<td>928</td>
</tr>
<tr>
<td>31</td>
<td>13</td>
<td>540</td>
</tr>
<tr>
<td>31</td>
<td>12</td>
<td>841</td>
</tr>
</tbody>
</table>

![Graph showing sensor data over time](image-url)
Outline

- TinyDB
- **Acquisitional Language Features**
  - Events
  - Buffers
  - Rates & Lifetimes
- **Acquisitional Processing (a taste)**
  - Optimization of selections
  - Buffering results
    » Choosing where to place storage
Event Based Processing

• **ACQP** - want to initiate queries in response to events

```sql
CREATE BUFFER birds(uint16 cnt)
   SIZE 1

ON EVENT bird-enter(...) 
   SELECT b.cnt+1
   FROM birds AS b
   OUTPUT INTO b
   ONCE
```

In-network storage

Subject to optimization
ON EVENT bird_detect(loc) AS bd
   SELECT AVG(s.light), AVG(s.temp)
   FROM sensors AS s
   WHERE dist(bd.loc,s.loc) < 10m
   SAMPLE PERIOD 1s for 10

[Coming soon!]
Event Based Processing

Time v. Current Draw

Event Based Trigger

Polling Based Trigger
Lifetime Queries

• Lifetime vs. sample rate

SELECT ...
LIFETIME 30 days

SELECT ...
LIFETIME 10 days
MIN SAMPLE INTERVAL 1s

Implies not all data is transmitted
(Single Node) Lifetime Prediction

Predicted Voltage vs. Actual Voltage (Lifetime Goal = 24 Wks)

Battery Voltage (ADC Units)

Time (in Hours)

Linear Fit (r = -0.92)
Actual Data
Predicted Lifetime

Insufficient Voltage to Operate (V=350)
Processing Lifetimes

• At root
  - Compute SAMPLE PERIOD that satisfies lifetime
  - If it exceeds MIN SAMPLE PERIOD (MSP), use MSP and compute transmission rate

• At other nodes – use root’s values or less

• Root = bottleneck
  - Multiple roots?
  - Adaptive roots?
In-network Buffers

- In-network storage needed for:
  - Offline delivery / high sample rates
  - Result correlation (joins)
  - Power conservation

```sql
CREATE TABLE myLight SIZE 5 (id uint16, value uint16)

SELECT nodeid, light INTO myLight
SAMPLE PERIOD 100ms

SELECT WINMAX(5, light) FROM myLight
SAMPLE PERIOD 500ms
```
Outline

• TinyDB

• Acquisitional Language Features
  – Events
  – Buffers
  – Rates & Lifetimes

• Acquisitional Processing (a taste)
  – Optimization of selections
  – Buffering results
    » Choosing where to place storage
Declarative -> Optimizable

• Queries *don’t* specify:
  - Where operators run
  - Order in which operators run
  - What algorithm operators use
  - Duty cycles
  - Rates, in lifetime queries
  - Path along which data is routed

  ...

• Easy to express, power-efficient, and fault-tolerant!
  - Through optimizations!
SELECT light, mag
FROM sensors
WHERE pred1(mag)
AND pred2(light)
SAMPLE INTERVAL 1s

- Energy cost of sampling mag >> cost of sampling light
  - 1500 uJ vs. 90 uJ
- Correct ordering (unless pred1 is very selective):
  1. Sample light  Sample mag  Sample light
  2. Sample light  Apply pred2  Apply pred1
  3. Apply pred1  Sample mag  Sample light
  4. Apply pred2  Apply pred1  Apply pred2

At 1 sample / sec, total power savings could be as much as 4mW, same as the processor!
Optimizing in ACQP

- **Sampling** = “expensive predicate”
- **Subtleties**
  - Which predicate to “charge”?  
  - Sampling must precede some operators
- **Solution**
  - Treat sampling as a separate task
  - Build a partial order
  - Use series-parallel scheduling algorithm to find best schedule
  
Exemplary Aggregate Pushdown

SELECT WINMAX(light,8s,8s) FROM sensors WHERE mag > x SAMPLE INTERVAL 1s

Unless > x is very selective, correct ordering is:
Sample light
Check if it’s the maximum
If it is:
Sample mag
Check predicate
If satisfied, update maximum
Event-Join Duality

ON EVENT \( E(\text{nodeid}) \)
SELECT \( a \)
FROM sensors AS \( s \)
WHERE \( s.\text{nodeid} = e.\text{nodeid} \)
SAMPLE INTERVAL \( d \) FOR \( k \)

SELECT \( s.a \)
FROM sensors AS \( s \),
    events AS \( e \)
WHERE \( s.\text{nodeid} = e.\text{nodeid} \)
AND \( e.\text{type} = E \)
AND \( s.\text{time} - e.\text{time} < k \)
AND \( s.\text{time} > e.\text{time} \)
SAMPLE INTERVAL \( d \)

- **Problem**: multiple outstanding queries (lots of samples)
- **High event frequency** → **Use Rewrite**
- **Rewrite problem**: phase alignment!
- **Solution**: subsample
Placing Buffers

- Buffer location not specified by query
- TinyDB chooses where storage lives
  - Current implementation: partition by nodeid
  - Other options
    » At root
    » At storage rich node
    » At a randomly selected node (for load balancing)
  - Open problems:
    » Finding storage
    » Choosing the best location
    » Making storage fault-tolerant?
• **ACQP**: Focus on acquisitional issues
  - Acquisitional Language Features:
    » Events
    » Lifetimes
    » Buffers
  - Declarative interface enables transparent acquisitional optimizations:
    » Order of selections
    » Events $\Leftrightarrow$ Joins
    » Placement of buffers
  - Making TinyDB more efficient and robust than the average sensor network programmer!
Questions?
Attribute Driven Topology Selection

- **Observation:** internal queries often over local area*
  - Or some other subset of the network
    - E.g. regions with light value in [10,20]

- **Idea:** build topology for those queries based on values of range-selected attributes
  - For range queries
  - Relatively static trees
    - Maintenance Cost
Attribute Driven Query Propagation

```sql
SELECT ...
WHERE a > 5 AND a < 12
```

Precomputed intervals = Semantic Routing Tree (SRT)
Attribute Driven Parent Selection

Even without intervals, expect that sending to parent with closest value will help.

\[
[3,6] \cap [1,10] = [3,6] \\
[3,7] \cap [7,15] = \emptyset \\
[3,7] \cap [20,40] = \emptyset
\]
Simulation Result

Nodes Visited vs. Query Range

- **Best Case (Expected)**
- **Closest Parent**
- **Nearest Value**
- **Snooping**

Query Size as % of Value Range

(Random value distribution, 20x20 grid, ideal connectivity to (8) neighbors)
ACQP

• How does the user control acquisition?
  - Rates or lifetimes.
  - Event-based triggers

• How should the query be processed?
  - Sampling as an operator!
  - Events as joins

• Which nodes have relevant data?
  - Semantic Routing Tree
    » Nodes that are queried together route together

• Which samples should be transmitted?
  - Pick most “valuable”? 
Delta Encoding

- Must pick most valuable data
- How?
  - Domain Dependent
    » E.g., largest, average, shape preserving, frequency preserving, most samples, etc.
- Simple idea for time-series: order biggest-change-first
Choosing Data To Send

- Score each item
- Send largest score
  - Out of order -> Priority Queue
- Discard / aggregate when full
Choosing Data To Send

Time vs. Value

[1,2]
Choosing Data To Send

| 2-6 | = 4 |
| 2-15 | = 13 |
| [2,6] | [3,15] | [4,1] |
Choosing Data To Send

[1,2]  [3,15]

|2-6| = 4  |15-4| = 11

Time vs. Value

Value

Time

0 2 4 6 8 10 12 14 16

1 2 3 4
Choosing Data To Send

Time vs. Value

[1,2] [3,15] [4,1] [2,6]
Choosing Data To Send

![Graph showing time vs. value with intervals: [1,2], [2,6], [3,15], [4,1]]

- Time vs. Value
  - Time: 1, 2, 3, 4
  - Value: 0, 2, 4, 6, 8, 10, 12, 14, 16

- Intervals: [1,2], [2,6], [3,15], [4,1]
Delta + Adaptivity

- 8 element queue
- 4 motes transmitting different signals
- 8 samples /sec / mote
Aggregate Prioritization

• **Insight:** Shared channel enables nodes to hear neighbor values

• **Suppress values that won’t affect aggregate**
  - E.g., MAX
  - Applies to all exemplary, monotonic aggregates e.g. top/bottom N, MIN, MAX, etc.
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
Simulation: Aggregate Prioritization

- 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

![Graph Showing Messages/Epoch vs. Network Diameter](image)
ACQP Summary

• Lifetime & event based queries
  - User preferences for *when* data is acquired

• Optimizations for
  - Order of sampling
  - Events vs. joins

• Semantic Routing Tree
  - Query dissemination

• Runtime prioritization
  - Adaptive rate control
  - *Which* samples to send
Fun Stuff

• Temporal aggregates

• Sophisticated, sensor network specific aggregates

• Delta compression, wavelets, etc.
Temporal Aggregates

• TAG was about “spatial” aggregates
  - Inter-node, at the same time
• Want to be able to aggregate across time as well
• Two types:
  - Windowed: $AGG(size, slide, attr)$
  - Decaying: $AGG(comb\_func, attr)$

- Demo!
Isobar Finding
Summary

• **Declarative queries are the right interface for data collection in sensor nets!**
  - Easier, faster, & more robust

• **Acquisitional Query Processing**
  - Framework for addresses many new issues that arise in sensor networks, e.g.
    » Order of sampling and selection
    » Languages, indices, approximations that give user control over *which data enters the system*

TinyDB Release Available - [http://telegraph.cs.berkeley.edu/tinydb](http://telegraph.cs.berkeley.edu/tinydb)