TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks

Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong
Outline

- Why TAG?
- Declarative Query
- In Network Aggregation
- Evaluation & Optimization
- Conclusion & Discussion
Sensor Networks

- Sensor Network is all about Data Querying

**Environment Sensing:** Earthquake detection

**Habitat Monitoring:** What are my cats doing when I am not at home.....
Why TAG?

• Challenges for Querying Data
  – Programming/Debugging is a nightmare
Why TAG?

• Challenges for Querying Data
  – Programming/Debugging is a nightmare
  – Have to take care of low level details

**Application 1** have to take care of these
Routing, Radio Contention, data storage and fault tolerance

**Application 2** have to take care of these **AGAIN!**
Routing, Radio Contention, data storage and fault tolerance
Why TAG is needed?

- Challenges for Collecting Data
  - Programming/Debugging is a nightmare
  - Have to take care of low level details

APP1:
SELECT MAX(mag)
FROM sensors
WHERE mag > thresh
EPOCH DURATION 64ms

APP2:
SELECT AVG(light), temp/10
FROM sensors
GROUP BY temp/10
EPOCH DURATION 1s

TAG: Allowing declarative queries by SQL-like APIs and do query optimization
A Layer of Indirection

Low Level Details
Why TAG?

- Challenges for Collecting Data:
  - Life time of a sensor is all about the energy.
  - Energy Consumption is all about the Radio.

One Instruction $\times 8000 =$ One Bit of Transmission
Why TAG?

• Challenges for Collecting Data
  – Life time of a sensor is all about energy
  – Energy Consumption is all about Radio
  – **Reduce** the amount of message transmission with In-network aggregation (Tiny Aggregation)
  – Exploit the semantic of SQL queries to reduce the amount of radio transmission
Outline

- Why TAG?
- Declarative Query
- In Network Aggregation
- Evaluation & Optimization
- Conclusion & Discussion
Declarative SQL-like Query

- Support Full Fledged SQL query

```sql
SELECT roomNo, AVG(light) 
FROM sensors 
GROUP BY roomNo 
HAVING AVG(light) > 200 
EPOCH DURATION 5s
```

<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>

Epoch DURATION: 5s
Declarative SQL-like Query

- Support Full Fledged SQL query and More

```sql
SELECT roomNo, AVG(light)
FROM sensors
GROUP BY roomNo
HAVING AVG(light) > 200
```

<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>
**Declarative SQL-like Query**

- **Support Full Fledged SQL query and More**

```
SELECT roomNo, AVG(light)
FROM sensors
GROUP BY roomNo
HAVING AVG(light) > 200
EPOCH DURATION 5s
```

<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>

```
SELECT {agg(expr), attrs}
FROM sensors
WHERE {selPreds}
GROUP BY {attrs}
HAVING {having Preds}
EPOCH DURATION i
```

- **MAX, MIN, COUNT,SUM, AVG, MEDIAN, COUNT DISTINCT, HISTOGRAM**
Outline

- Why TAG?
- Declarative Query
- In Network Aggregation
- Evaluation & Optimization
- Conclusion & Discussion
In Network Aggregation

- An Example:
  - Tree-like Topo & Level based Routing

![Diagram of a tree-like topology with levels and a query path](image-url)
In Network Aggregation

• An Example:
  – Tree-like Topo & Level based Routing
In Network Aggregation

- An Example:
  - ‘Global’ Synchronized Transmission

Figure 1: Partial state records flowing up the tree during an epoch.
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (without TAG)

Total Messages: 0
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (without TAG)

  Total Messages: 1
  Numbers: [5]
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (without TAG)

Total Messages: 6
Numbers: [5,7,4]
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (without TAG)

Total Messages: 10
Numbers: [5,7,4,8,9]
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (without TAG)

  Total Messages: 14
  Numbers: [5,7,4,8,9,3,1]
  MAX=9
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (with TAG)

Total Messages: 3
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (with TAG)

Total Messages: 7
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (with TAG)

Total Messages: 9
In Network Aggregation

• An Example:
  – SELECT MAX(temp) FROM sensors (with TAG)

Total Messages: 9 (vs 14)
MAX=9 delivered

Slot 3
Epoch

[8,9,5]
In Network Aggregation

• Formal Definition

\[ \text{Agg}_n = \{ f_{\text{init}}, f_{\text{merge}}, f_{\text{evaluate}} \} \]

\[ f_{\text{init}} \{ a_0 \} \rightarrow \langle a_0 \rangle \]

\[ F_{\text{merge}} \{ \langle a_1 \rangle, \langle a_2 \rangle \} \rightarrow \langle a_{12} \rangle \]

\[ F_{\text{evaluate}} \{ \langle a_1 \rangle \} \rightarrow \text{aggregate value} \]

**Example: Average**

\[ \text{AVG}_{\text{init}} \{ v \} \rightarrow \langle v, 1 \rangle \]

\[ \text{AVG}_{\text{merge}} \{ \langle S_1, C_1 \rangle, \langle S_2, C_2 \rangle \} \rightarrow \langle S_1 + S_2, C_1 + C_2 \rangle \]

\[ \text{AVG}_{\text{evaluate}} \{ \langle S, C \rangle \} \rightarrow \frac{S}{C} \]
In Network Aggregation

- Not All Operations can benefit from TAG
- Different Operations benefit from TAG differently
- Do further optimization based on different property

<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>Duplicate Sensitivity</td>
<td>MIN: dup. insensitive, AVG: dup. sensitive</td>
<td>Routing Redundancy</td>
</tr>
<tr>
<td>Exemplary vs. Summary</td>
<td>MAX: exemplary COUNT: summary</td>
<td>Applicability of Sampling, Effect of Loss</td>
</tr>
<tr>
<td>Monotonic</td>
<td>COUNT: monotonic AVG: non-monotonic</td>
<td>Hypothesis Testing, Snooping</td>
</tr>
</tbody>
</table>
Outline

- Why TAG?
- Declarative Query
- In Network Aggregation
- Evaluation & Optimization
- Conclusion & Discussion
Evaluation & Optimization

- Simulation Based Evaluation
  - Benefit From TAG varies
  - 50*50 Grid
  - No-loss link
  - Simple Topo

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

- Aggregation Functions: EXTERNAL, MAX, AVERAGE, COUNT, MEDIAN
- Y-axis: Total Bytes Xmitted
- X-axis: Aggregation Function
Evaluation & Optimization

• Optimization: Snooping
  – Utilize the Shared Channel to further reduce data transmission
Evaluation & Optimization

- Optimization: Hypothesis & Test
  - Use some blind/statistical/subset guess to further reduce transmission
• Optimization: Multiple Parents
  – Increased Reliability
    • Duplicate insensitive aggregates
    • Aggregates that can be expressed as a linear combination of parts
Evaluation & Optimization

• Evaluation: Effect of Loss
  – Single loss

(a) Maximum Error  (b) Average Error
Evaluation & Optimization

- Optimization: Child Cache
  - Increased Availability
    - Use old results when new results are not available
Evaluation & Optimization

- Optimization: Child Cache
  - Increased Availability
    - Use old results when new results are not available

Figure 8: Comparison of Centralized and TAG based Aggregation Approaches in Lossy, Prototype Environment Computing a COUNT over a 16 node network.
Conclusion

• Contributions:
  – Declarative Query Framework for Sensor Networks
  – In Network Aggregation and Optimization based on Semantics

• Comments/Discussion Questions:
  – Time Synchronization in Epoch
  – Tree Based Topo is failure prone
  – Real Deployment
  – Malicious Nodes
Camdoop: Exploiting In-network Aggregation for Big Data Applications

Austin Donnelly, Antony Rowstron, and Greg O’Shea

Presented by Mo Dong
Why Camdoop

- Shuffle Phase Involved All to All traffic
Why Camdoop

• Possible to Reduce Intermediate Data In Network
What is Camdoop

• **Reduce Amount of Traffic using Camcube Topo**
Synopsis Diffusion for Robust Aggregation in Sensor Networks

Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson

Presented by Hongyang Li
Motivation

• Problem with too much reliance on routing structure
  • Dynamic network condition due to node or link failure
  • High maintenance overhead
  • Difficult to find the best routing structure for all network conditions
• Synopsis Diffusion Approach
  • Less reliance on routing structure (though a good structure still helps)
  • Aggregate computation is more involving
Synopsis Diffusion

• Less reliance on routing
  – Communication order doesn’t matter: sensor A could report either before or after sensor B
  – Duplicate reading doesn’t matter: sensor A can send 1 or 10 copy of its reading and the final answer remains the same

• Synopsis Diffusion is Order- and Duplicate-Insensitive (ODI)
Synopsis Diffusion: 3 operations

- SG(r): generate a synopsis from sensor reading r
- SF(s1,s2): combine two synopses s1 and s2
- SE(s): evaluate synopsis s into final answer
Synopsis Diffusion: ODI

(a) Aggregation DAG

(b) Canonical left-deep tree
Check ODI

- $SG(r_1) = SG(r_2)$ for duplicate readings $r_1, r_2$
- $SF(s_1,s_2) = SF(s_2,s_1)$
- $SF(s_1, SF(s_2,s_3)) = SF(SF(s_1,s_2),s_3)$
- $SF(s,s) = s$
Example: (Inefficient) Count

• Suppose there are N nodes, each voting either 0 or 1. We want to count how many nodes have voted 1.

• Synopsis s: N-bit array

• SG(r) of node i: set the i-th bit to 1 iff r > 0; set all other bits to 0.

• SF(s1,s2): s1 OR s2

• SE(s): count the number of 1’s in s
Example: Probabilistic Counting

• Suppose each node voting 1 generate a random number.
  – The probability that the binary representation of random number ends with “0” is 1/2
  – The probability of ending with “00” is 1/4

• Think in the other direction
  – If some node outputs a binary random number ending with “0”, then with high probability there are about 2 nodes.
  – If some node outputs a binary random number ending with “00”, then with high probability there are about 4 nodes.

• Conclusion: the number of nodes is proportional to $2^i$, where $i$ is the length of the longest “0” tails
Example: Probabilistic Counting

• Uses \( k > \log(N) \)-bit array as synopsis instead of \( N \)-bit array
  – \( \text{CT}(x) \): toss a fair coin for maximally \( x \) times, output the first time that heads came up, or \( x \) if no heads came up.
  – \( \text{SG} \): Output a bit vector of length \( k \) with only \( \text{CT}(k) \) bit set
    • Set bit 1: I see a trail of “0”
    • Set bit 2: I see a trail of “00”
    • Set bit 3: I see a trail of “000”
  – \( \text{SF}(s_1, s_2) \): \( s_1 \) OR \( s_2 \)
  – \( \text{SE}(s) \): For the smallest \( i \) such that \( s[i]=0 \), output \( 2^{(i-1)}/0.77351 \)
Example: Uniform Sampling

• Synopsis: K tuples <value, rand, id>
• SG: each node $i$ generate a random number and construct $<value_i, rand_i, i>$
• SF(s1, s2): keep the K tuples <value, rand, id> with the highest rand
• SE(s): output all the value fields in <value, rand, id>
Uniform Sampling is ODI

- $\text{SG}(r_1) = \text{SG}(r_2)$ for duplicate $r_1, r_2$: trivial
- $\text{SF}(s_1, s_2) = \text{SF}(s_2, s_1)$
  - the $K$ tuples with highest rand in $s_1$ and $s_2$
- $\text{SF}(s_1, \text{SF}(s_2, s_3)) = \text{SF}(\text{SF}(s_1, s_2), s_3)$
  - the $K$ tuples with highest rand among $s_1, s_2, \text{ and } s_3$
- $\text{SF}(s, s) = s$
  - the $K$ tuples with highest rand in $s$
Implicit Acknowledgement

• Suppose A sends its reading to B for aggregation. How does A know that B has received its reading?

• Explicit ack: waste energy/bandwidth

• Implicit ack:
  – A sends x
  – B aggregates x with y and obtains \( z = SF(x,y) \)
  – We must have \( x \leq z \) (\( \leq \) as defined in the context)
Routing Structure: Ring
Routing Structure: Adaptive Ring

- Node can use implicit ack to check whether transmission is successful
- In case of unsuccessful transmission
  - Retransmit: waste energy/bandwidth/time
  - Adaptive Ring: choose another set of parents with better connectivity
Routing Structure: Adaptive Ring

• When to switch to a new ring?
  • If the number of times that any node in my parent ring retransmits my synopsis falls below a threshold
• How to switch to a new ring?
  • Wake up at the corresponding time slot
• Which ring to switch to?

<table>
<thead>
<tr>
<th>Ring</th>
<th>Overheard Transmissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-1</td>
<td>Overhear 3 transmissions</td>
</tr>
<tr>
<td>Current</td>
<td>Overhear 4 transmissions</td>
</tr>
<tr>
<td>i+1</td>
<td>Overhear 5 transmissions</td>
</tr>
<tr>
<td>i+2</td>
<td>Overhear 6 transmissions</td>
</tr>
</tbody>
</table>

• More transmissions overheard as ring depth i increases: moving down is probably a good choice
• After moving to ring i+1, the node has better connectivity with its parents
Table III. Comparison of Aggregation Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>% Nodes</th>
<th>Error (uniform)</th>
<th>Error (skewed)</th>
<th>Error (Gaussian)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG</td>
<td>&lt; 15%</td>
<td>0.87</td>
<td>0.99</td>
<td>0.94</td>
</tr>
<tr>
<td>TAG2</td>
<td>N/A</td>
<td>0.85</td>
<td>0.98</td>
<td>0.92</td>
</tr>
<tr>
<td>GOSSIP</td>
<td>N/A</td>
<td>0.91</td>
<td>0.99</td>
<td>0.93</td>
</tr>
<tr>
<td>RINGS</td>
<td>65%</td>
<td>0.33</td>
<td>0.19</td>
<td>0.21</td>
</tr>
<tr>
<td>ADAPT. RINGS</td>
<td>95%</td>
<td>0.15</td>
<td>0.16</td>
<td>0.15</td>
</tr>
<tr>
<td>FLOOD</td>
<td>≈ 100%</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
</tr>
</tbody>
</table>
Evaluation
Conclusion

Reliance on routing structure

TAG

Synopsis Diffusion

Complex aggregate computation
Discussion Points

• Can every aggregation task be written in an ODI style?
• Is aggregation error always a bad thing?