TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks
Samuel Madden, Michael J Franklin, Joseph M Hellerstein, Wei Hong

Pradeep Kumar Gunda
(Thanks to Jigar Doshi and Shvnath Babu for some slides)
Sensor Networks used for monitoring in various fields
- Civil engineers to monitor buildings during earthquakes
- Biologists for habitat monitoring
People prefer summary reports not individual values

**Aggregation common to all these applications!**

Must be a core service and easy to use.
TAG fills this void
Before TAG

- Centralized approach
  - Transfer everything to base station
  - No suppression – high energy usage, traffic

- Directed Diffusion
  - Viewed aggregation as a application-specific operation
  - Aggregation API in routing layer
  - No declarative query language like TAG
  - Not for any generic aggregation operators
What is TAG

- Tiny Aggregation for Sensor Networks
- SQL – like interface eg. Min, Max, Count
- Sensitive to constraints of ad-hoc sensor networks
- Query inserted into network over an existing routing protocol
- Aggregation done along the reverse path

- Combines the research in **networking** community with **database** community
DBMS in a nutshell

Select max(wins) from Basketballwins
Where year='2002'
Group by tournament

<table>
<thead>
<tr>
<th>Team</th>
<th>Year</th>
<th>Wins</th>
<th>Tournament</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duke</td>
<td>2002</td>
<td>25</td>
<td>ACC</td>
</tr>
<tr>
<td>Duke</td>
<td>2003</td>
<td>35</td>
<td>NCAA</td>
</tr>
<tr>
<td>UNC</td>
<td>2002</td>
<td>20</td>
<td>ACC</td>
</tr>
<tr>
<td>UNC</td>
<td>2003</td>
<td>25</td>
<td>NCAA</td>
</tr>
</tbody>
</table>
Data Management System

High-level Query Q

DBMS

Answer

Translates Q into best execution plan for current conditions, runs plan

Keeps data safe and correct despite failures, concurrent updates, online processing, etc.
Data Streams

User/Application

Register
Continuous Query
(Standing Query)

Stream Query Processor

Result

Input streams
What can DBMS offer for Sensor Networks??

- Express aggregation as SQL
- *Specify what you want. Not how to get*
- Users need not write low level programming language code!!!
  - Less bugs
  - Don’t worry about optimization

- Techniques from parallel/distributed db
- Sensor network is a *stream of sensor* readings to base station
Query Model

One Table: sensors

SELECT AVG(volume), room FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(VOLUME) > threshold
EPOCH DURATION 30s

In general
SELECT \{agg(expr), attrs\} FROM sensors
WHERE \{selPreds\}
GROUP BY \{attrs\}
HAVING \{havingPreds\}
EPOCH DURATION /

Difference between TAG & SQL: Continuous Output
Aggregate Structure

- Standard SQL supports “the basic 5”: MIN, MAX, SUM, AVERAGE, and COUNT

- TAG supports any function conforming to
  - **Initializer i**: Instantiates a record for a single sensor value
  - **Merging function f**: Merges two partial state records
  - **Evaluator e**: Computes the actual value of the aggregate from a partial state record

**Example - average**
- \( i\{v\} \rightarrow <v,1> \)
- \( f\{<S_1, C_1>, <S_2, C_2>\} \rightarrow <S_1 + S_2, C_1 + C_2> \)
- \( e\{<S_1, C_1>\} \rightarrow S_1/C_1 \)

- TAG supports MEDIAN, HISTOGRAM and COUNT DISTINCT also
Classifying Aggregates

- **Duplicate Sensitive** (yes/no)
- **Exemplary/Summary**
- **Monotonic**
  
  \[ s' = f(a,b) \]
  
  \[ e(s') \geq \max(e(s1),e(s2)) \quad \text{OR} \]
  
  \[ e(s') \leq \min(e(s1),e(s2)) \]
  
  Decides whether predicate can be applied in network

- **Partial State**
  
  - Distributive (partial state’s size same as final aggregate)
  - Algebraic (partial states are not themselves aggregate)
  - Holistic (No useful partial aggregation)
  - Unique
  - Content Sensitive
# Aggregate Taxonomy

<table>
<thead>
<tr>
<th></th>
<th>MAX, MIN</th>
<th>COUNT, SUM</th>
<th>AVERAGE</th>
<th>MEDIAN</th>
<th>COUNT DISTINCT</th>
<th>HISTOGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate Sensitive</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Exemplary (E), Summary (S)</td>
<td>E</td>
<td>S</td>
<td>S</td>
<td>E</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Monotonic</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Partial State</td>
<td>Distributive</td>
<td>Distributive</td>
<td>Algebraic</td>
<td>Holistic</td>
<td>Unique</td>
<td>Content-Sensitive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partial state size</th>
<th>Distributive</th>
<th>Algebraic</th>
<th>Holistic</th>
<th>Unique</th>
<th>Content-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Same as final aggregate</td>
<td>Constant</td>
<td>Data proportional</td>
<td>Distinct value proportional</td>
<td>Proportional to some data property</td>
</tr>
</tbody>
</table>
Requirements of the Routing Algorithm

- Deliver Query requests to all nodes
- Route from every node to root
- No Duplicates! *(Affects some aggregates like count, avg)*
  - Does it violate the end-to-end principle?
- A simple example proposed: tree based routing
Tree Based Routing

- One root
- Any interior node sets sender as parent and sets its level to that of parent + 1
- Rebroadcasts
- Message sent by node to its parent eventually reaches root
- Reselect parent after k silent epochs
The TAG Algorithm

2 Phases

- **Distribution**: Queries are pushed down the network.
  - Parents broadcast queries to their children

- **Collection**: Aggregate values continuously sent from children to parents
  - Reply from all children required before forwarding an aggregate value
    - TDMA like partitioning
    - Children must deliver records during a parent-specified time interval

- Parent collects all values (including its own) and sends the aggregate up the tree
Flow of partial State

- Parent reception interval must be chosen carefully
- All children must be able to report
- Cannot exceed end of epoch
- However we can always make the algorithm pipelined
Pipelined Aggregation

SELECT COUNT(*)
FROM sensors
Pipelined 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>Epoch #</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Epoch 4
Piped Aggregation

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

- Simple aggregation mechanism
- Complicated by HAVING clause
- Group eviction to solve storage problem
- Evicted tenant sent to parent

SELECT AVG(light), temp/10
FROM sensors
GROUP BY temp/10
Simulation Environment

Java-based simulation & visualization for validating algorithms, collecting data.

Sensors arranged on a grid, radio connectivity by Euclidian distance

Communication model

- Lossless: All neighbors hear all messages
  - Symmetric links
  - No collisions, hidden terminals, etc.

- Realistic
  - Number of hops related to distance but not proportional
Simulation Results

- **2500 nodes, d=50**
- **TAG outperforms centralized approach by an order of magnitude in most cases**
  - Does equally well in the worst case
  - Actual benefit depends on the topology
Optimizations

🌿 Snooping
- Overhear packets – can initiate aggregation if missed
- Can also be used for suppression!

🌿 Hypothesis testing
- Guess the value of aggregate & suppress
- *Send only if current val > MAX*
- Can be applied to a variety of aggregates such as MAX
Experiment: Hypothesis Testing

Uniform Value Distribution, MAX Query

Messages/Epoch vs. Network Diameter

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

Messages / Epoch

Network Diameter

No Guess
Guess = 50
Guess = 90
Snooping

Uniform Value Distribution, MAX Query
TAG Loss Tolerance

- Maintain a list of the link signal quality to the neighbors and shift to better if available.

- Pick a new parent if no hello for a $\tau$.
  - You can pick a node below you in the tree so child may have to reselect their parent.
Experiment: Effects of Loss

Percent Error From Single Loss vs. Network Diameter

Network Diameter

Percent Error From Single Loss

AVERAGE
COUNT
MAX
MEDIAN
Experiment: Benefit of Cache

Percentage of Network Involved vs. Network Diameter

- No Cache
- 5 Rounds Cache
- 9 Rounds Cache
- 15 Rounds Cache
Summary

• TAG is based on a declarative interface
  • Makes network tasking easier for the end user

• TAG outperforms centralized approaches in most cases

• Relies heavily on underlying routing layer
  • Placement of query tree constrained by characteristics of routing tree
Critique

- Fault Tolerance – too simplistic
  - How high is the failed node on the tree
- Mobility
- Multiple Queries?
- Generic?
- Multiple sinks / Sink mobility?
- More hypothesis testing - more problems!
- Compression + aggregation?
- Energy budgeting?
Questions ?