TAG & Willow

Aggregates in P2P Network

Within Physical or Virtual Hierarchy

Overview

- Consider SQL aggregates
  - Select class, SUM(grade), AVERAGE(grade), COUNT(grade)
  - From STANFORD_CLASS_STUDENT_GRADES
  - Where class Like 'CS%'
  - Group by class
  - CS345, 80.0, 4.0, 20
  - CS240, 60.0, 2.0, 30
  - How to do this in a distributive fashion?
  - Hierarchy – physical or virtual

Agenda

- Tiny Aggregation – TAG (Madden et al)
  - Background
  - TAG Features
  - TAG overview
  - Environment
  - Query model
  - TAG Implementation
  - Optimization & Loss mitigation
  - Simulation & Prototype
  - Comments

Agenda

- Willow (Renesse et al)
  - Willow overview
  - Willow model
  - Operations
  - Willow implementation
  - Simulation
  - Comments

TAG Background

- Sensor network
- Berkeley experiments
  - http://robotics.ics.berkeley.edu/~gaster/29Palm0103/

TAG Features

- SQL style interface for aggregates
- Efficiently distribute and execute aggregates IN NETWORK
- Time and power efficiency
- Work well in low power wireless network
Basic TAG Operations
- Routing tree rooted at base station
- Query posted from base station
- Query piggybacked on network protocol
- Sensors route back data to the root through the same routing tree
- Aggregate happens at each node as data flows back

TAG Environment
- Motes hardware/software environment
  - Mote: computer sensors radio power
  - CSMA-like MAC with random back off
  - Unreliable communication
  - Battery powered (transmitting dominates)
- Routing tree
  - Deliver query requests to all nodes
  - Route from node to root (no duplicate message)

TAG Environment (Continued)
- Build routing tree

TAG Environment (Continued)
- Build routing tree

TAG Environment (Continued)
- Build routing tree

SQL-Style Query Model
- One 'Sensor' table at each mote
- One attribute (column) per input (e.g. temperature, light, room)
- Example
  SELECT AVG(volume).room FROM sensors
  WHERE floor = 6
  GROUP BY room
  HAVING AVG(volume) > threshold
  EPOCH DURATION 30s
SQL/Style Query Model (Cont’d)

- General syntax
  
  ```sql
  SELECT [agg(expr)], [atts]
  FROM sensors
  WHERE [setPreds]
  GROUP BY [atts]
  HAVING [havingPreds]
  EPOCH DURATION
  
  Q: What can be done locally?
  ```

- Epoch
  
  - Stream of data
  - Aggregate readings in the same time interval
  - Wait for child motes gathering and transmitting data
  - Lower bound determined by local processing (30 ms)

SQL/Style Query Model (Cont’d)

- How to aggregate?
  
  - Merging function f: \( \mathbb{R}^m \rightarrow \mathbb{R} \)
    - e.g., merging function for AVERAGE
      
      - partial state: \( \{ \text{SUM}, \text{COUNT} \} \)
      - \( \{ \text{SUM}(x), \text{COUNT} \} \)
        - \( \{ \text{SUM}(x) + y, \text{COUNT} \} \)
        - Initializer: initial state value from a single sensor (e.g., \( \{x, 1\} \) for AVERAGE)
        - Evaluator e: compute aggregation from partial state (e.g., \( \text{SUM}/\text{COUNT} \) for AVERAGE)

- What aggregate can be executed in TAG?
  
  - \( \text{COUNT, MIN, MAX, SUM, AVERAGE} \)

- General classification:
  
  - Duplicate sensitivity: e.g. \( \text{COUNT} \) vs. \( \text{COUNT} \text{DISTINCT} \)
  - Exemplary/Summary: Sensitive to loss or not
  - Monotonic: For any \( x, y \), either \( e(x) \geq e(y) \) or \( e(x) \leq e(y) \)
  - MIN(e(x), e(y))
  - Partial state: performance inversely related to amount of partial state
    - Distributive: Algebraic
    - Content sensitive: Unit of
    - Holistic

Implement Aggregates

- Naively, aggregate at base
- Basic TAG – distribution and collection
  
  - Distribution: Set time interval for children to deliver partial state
  - Collection: power up small portion of epoch when data flows back

Table 1: Classes of aggregates

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Partial</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Partial state record flowing up the tree during an epoch
Implement Aggregates (Cont’d)

- Basic TAG (Continued)
  - How to select receiving time interval?
  - Too short, not reaching deep nodes.
  - Too long, exceeds epoch. And consumes more power.
  - Epoch (sample rate) lower bound (iLevel)
  - Pipelining to increase sample rate (delay aggregates) ??

Implement Aggregates (Cont’d)

- TAG with grouping
  - Push grouping expression (attribute) down
  - Nodes choose group
  - Update aggregate per group (group ID)
  - Rehire more storage and transmission
  - Evict and forward
  - When can HAVING be restricted in network?
    - Monotonic
      - HAVING MAX(Attr) □(or MIN(Attr) □)

Implement Aggregates (Cont’d)

- TAG with grouping (Continued)

Figure 2: A sensor network (500 with an in network, grouped aggregate applied to it right). Parenthesized numbers represent nodes that contribute to the average

Implement Aggregates (Cont’d)

- TAG advantages
  - Decrease communication requirements
  - Tolerate loss and disconnection
  - Reduce load for node up in the tree
  - Power efficient
  - Increase sample rate (less transmission time and collision)

Evaluation by Simulation

- 3 communication models
- 50/30 motes
- Evaluate amount of data transmitted

Evaluation by Simulation (Cont’d)

- Centralized vs. TAG
  - Centralized not sensitive to class of aggregates
  - Order of magnitude gain
  - Why these numbers? 5000 bytes/epoch for count, 10,000 for average, 90,000 for median.
  - COUNT DISTINCT (3,000)
  - Topology affects benefit
  - Grouping performance not sensitive to eviction policy

(a) Simple
(b) Random
(c) Realistic

Figure 3: The TAG Simulator, with Three Different Communications Models, Diameter = 26
Prototype
- TAG accuracy much better than centralized approach.
- Why? Much less radio link contention

Optimization
- Leverage shared radio channel – snooping
  - Find out missing initial retrans from neighboring motes’ aggregating message
  - Reduce transmission for certain aggregates (e.g., MAX-MIN)
- Generalized Hypothesis testing
  - Mote decides contributing to aggregate or not based on a guess
  - Applicable to monotonic, exemplary aggregates (e.g., MIN, MAX) – root get MIN for first level, then send new query with this MIN
  - For summary aggregates, send error bound with query

Mitigate Loss
- Shift to a different parent if link quality degrades to a certain level.
- Single node loss -> recovery time proportional to depth of the lost mote
- Error sensitive to aggregate class, where is the lost node, network size, data distribution

Mitigate Loss (Cont’d)
- Child caching
- Multiple parents

Comments
- Physical tree has single point of failure. How about multiple trees rooted at different motes?
- What if power consumption is not concern?
- Any advantage of centralized aggregates? More robust against malicious mote?
- Can we compute variance, standard deviation, etc.?
Willow model

- Each agent has a 128-bit binary ID
- Virtual binary tree with tree depth 129 (0 left, 1 right)
- Domain and members
- Domain attributes
  - Leaf nodes (agents’ physical data)
  - Internal nodes – aggregate function
- Aggregate recalculation
  - Leaf attribute, membership, aggregate function change
  - Quick convergence but no instantaneous consistency guarantee (What does this mean?)

Fig. 1. Willow tree with four agents using three bit identifiers. The maximum load in each domain is indicated between braces.

Willow Operations

- DHT routing
- Monitoring
  - Query installed from one agents, propagated to others
  - Retrieve results in the form of (ID, value, count)
  - Stop when count approaches total number
- Selective multicast
  - Multicast based on monotonic aggregates
  - Query aggregated using “OR”
  - Simplify when too many terms – broadcast at upper level

Implementation

- Each agent maintains domain information for 128 domains along its path to root
- Domain information include
  - Domain ID: \( 128 \) digit prefix of agent ID
  - Friends in peer domain: members of \( \square \) (if agent is in \( \square \), member of \( \square \) otherwise). Think about Chord fingers.
  - Which member in peer domain? – (ademia)
  - In practice randomly adaptive single friend
  - Attributes of left and right child domain \( \square \), \( \square \)

Implementation (Cont’d)

<table>
<thead>
<tr>
<th>level</th>
<th>friend</th>
<th>child</th>
<th>contact</th>
<th>candidate</th>
<th>maxload</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>0</td>
<td>001</td>
<td>001</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>001</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>5</td>
</tr>
</tbody>
</table>

Fig. 2. Data maintained by agent 001 in Figure 1. In actuality, not the identifiers of other agents are stored, but their IP addresses and boot times.

Implementation (Cont’d)

- Multicast using friend links
  - Integer d in message indicate domain level
  - Multicast in domain level \( \square \)
  - Forward message to one of the friends
  - Selective multicast using aggregate
  - Hops \( O(\log(N)/\log(\log(N))) \) (Why?)
  - Different multicast tree for agents – load balancing
**Implementation (Cont’d)**

- Attribute propagation
  - How does agent derive aggregates at domain $\Box \bigcirc \Box$ from $\Box \bigcirc \Box$ and $\Box \bigcirc \Box$?
  - How does agent know aggregates that it does not belong to?
  - Domain contact election process
    - Result: 1. All agents are contact of exactly one internal domain, except for the oldest. 2. Older agent in charge of larger domain.
  - Domain contact sends attribute to peer domain
    - Child contact sends update to friend in $\Box \bigcirc \Box$
    - That friend multicast in $\Box \bigcirc \Box$

![Example of the zippering protocol](image)

**Evaluation by Simulation**

![Graph](image)

Fig. 5. Median latency as a function of the number of agents for different friend selection strategies. The error bars denote 95% confidence intervals.

**Evaluation (Cont’d)**

![Graph](image)

Fig. 6. Update latency as a function of the number of agents for different friend selection strategies.

**TAG Vs. Willow**

- Physical tree Vs. virtual tree
- Short distance links Vs. long haul links
- Single tree Vs. multiple trees
- Query from root Vs. query from any nodes