Storage and Processing II

Presented by –
Anjali Jhingran
Christopher Wells

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

Samuel Madden, Michael J. Franklin, and Joseph M. Hellerstein
U C Berkley
Wei Hong
Intel Research, Berkley

Goal

- Express simple declarative queries
- Effectively distribute and executes them in networks of low-power wireless sensors
- Show generic properties of aggregate that affect performance
- Show advantage of a TAG approach over traditional centralized, out-of-network methods

Motivation

- Often data extracted from sensor consists of summaries rather than raw readings
- Aggregate is central to emerging sensor network applications
  - It must be provided as a core service
- Instead of a set of extensible low level C APIs, service should consist of a generic, easily invoked high level programming abstraction
TAG

- A generic aggregation service for ad-hoc networks of TinyOS motes
- Consists of following attributes:
  - Provides simple declarative interface for data collection
  - Intelligently distributes and executes aggregate queries in sensor networks in a time and power efficient manner
  - Sensitive to resource constraints and lossy communication properties of wireless sensor networks

Aggregate Queries

- Supports SQL-style queries (without joins) over a single table called "sensor"
- Table can be thought of as an append-only relational table with one attribute per input of the motes
- Epoch is the duration of time a device waits before acquiring and transmitting each successive sample
- Aggregate is implemented via three functions:
  - Merging function
  - Initializer
  - Evaluator

Aggregate Queries

- Structure of Aggregate:
  - For example, \( <z> = f(<x>, <y>) \)
  - \( <x> \) and \( <y> \) are multi-valued partial state records
  - Each partial state record consists of a pair of values
  - \( f(<S_1, C_1>, <S_2, C_2>) = <S_1 + S_2, C_1 + C_2> \)

Advantages of TAG Over SQL-Query

- Primary difference between TAG query and SQL query is that output of TAG query is stream of values rather than a single aggregate value or batched results
- In monitoring applications, continuous results are more useful than a single aggregate
- It allows user to understand network behavior over time
- It allows user to observe transient effects (such as message loss) which would be hard to interpret otherwise
Taxonomy of Aggregates

Are classified according to four categories:

- **Duplicate**
  - Duplicate Insensitive
    - Remain unaffected by duplicate readings from a single device
  - Duplicate Sensitive
    - Aggregate will change when a duplicated reading is reported

- **Exemplary and Summary**
  - Exemplary
    - Returns one or more representative values from the set of all values
  - Summary
    - Computes some property over all values

**Partial State**

- **Distributive:** Simply the aggregate for the partition of data over which they are computed
- **Algebraic:** Partial state records are not themselves aggregate for the partitions
- **Holistic:** No useful aggregation can be done and all the data must be brought together by the evaluator
- **Unique:** Similar to Holistic, except the amount of state to be propagated is proportional to the number of distinct values in partition

**Content-sensitive**

- **Monotonic:** When two partial state records s1 and s2 are combined via f, the resulting state will have the property either for all s1, s2, e(s') = MAX (e(s1), e(s2)) or e(s') = MIN (e(s1), e(s2))

<table>
<thead>
<tr>
<th>Attribute</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1: Classes of Aggregates

**Attribute Catalog**

- Each mote has a small catalog of attributes
- Central query processor caches or stores the attributes of all the motes it may access
- Motes do not need to advertise their catalog of attributes
- All motes don’t require to have global knowledge of all attributes
- Attribute names are not carried with results
  - This leads to a significant reduction in amount of data that must be propagated with each tuple.
Tiny Aggregation Process

- **TAG consists of two phases:**
  - **Distribution Phase**
    - Aggregation queries are pushed down into the network
  - **Collection Phase**
    - Aggregate values are continually routed up from children to parent
- **GOAL:** use as few messages as possible
- **Mechanism:**
  - Collection phase must ensure that parents in the routing tree wait until they have heard from their children before propagating an aggregate value for the current epoch

**Mechanism**

- **p** receives a message from another mote or user
- **It awakens**
- **It synchronizes its clock according to timing information in the message**
- **Prepares to participate in the aggregation**
- **p chooses the sender of the message as its parent**
- **Aggregate query r includes the interval when the sender is expected to hear partial state record from p**
- **p then forwards r down the network, setting the delivery interval for the children to be slightly different**
- **Nodes continues to forward the request in this manner until the query has been propagated throughout the network**

**Grouping**

*Figure 1: Partial state records flowing up the tree during an epoch.*

*Figure 2: A sensor network (left) with an in-network, grouped aggregate applied to it (right). Paranormal numbers represent nodes that contribute to the average.*
Grouping

- Problem
  - Number of groups can exceed available storage on any one device

- Solution
  - Evict one or more groups from local storage
  - Evicted victim is sent to parent
  - Parent may hold it or forward it further up
  - Evicting partially computed groups is known as partial pre-aggregation

Benefits

- Its ability to dramatically decrease the communication required to compute an aggregate
- Ability to tolerate disconnections and loss
- Each mote is required to transmit only a single message per epoch, regardless of its depth in the tree
- By explicitly dividing time into epochs, a convenient mechanism for idling the processor is obtained
  - During an interval, radio and processor can be put down into deep sleep
- Properly compute aggregate for groups even when the amount of group information exceeds available storage on any device

Reduction in Communication

- Topology dependence of TAG
  - The benefits of TAG are more or less pronounced depending on the topology
    - In a flat single-hop environment where all motes are directly connected to root, TAG is no better than centralized network
    - When n motes are arranged in a line
      - Centralized approach takes $n^2/2$ partial state records to transmit
      - TAG takes $n$ partial state records to transmit
    - In worse case, (e.g. Median) TAG’s performance is equal to centralized approach
Optimization

- **Facts:**
  - Motes communicate over shared radio channel
  - Every message is effectively broadcast to all other nodes in the range

- **Aggregation by snooping**
  - A lost node when hears another device reporting an aggregate, it can assume it too should be aggregating
  - Snooping doesn’t require to listen all the time, it can be done at a predefined intervals,
  - Snooping can reduce number of messages sent for some class of aggregates

---

Optimization testing

- A node is presented with a guess as to the proper value of an aggregate
- It can decide locally whether contributing its reading and the readings of its children will affect the value of the aggregate

![Figure 5: Benefit of hypothesis testing for MAX](image)

---

Tolerance to loss

- Networking faults can be monitored and adapted at two levels
  - When a node $n$ observes that link quality to its parent $p$ is significantly worse than that of some other node $p'$, it chooses $p'$ as its new parent if
    - $p'$ is as close or closer to the root as $p$
    - $p'$ does not believe $n$ is its parent
  - When a node observes that it hasn’t heard from its parent (relative to epoch duration) it assumes its parent has failed or moved away
    - It resets its level (to infinity) and picks a new parent from neighbor table
    - This causes node to select a parent from its underneath sub-tree
    - Child nodes must reset their parent when they observe that their parent level has gone up

---

Tolerance to loss

- **Child cache**
  - Parent remember the partial state record their children reported for some number of rounds
  - It uses those previous values when new values are unavailable due to lost child messages

- **Drawbacks**
  - Temporarily smear the aggregate values that are computed
  - It uses memory that could be used for group storage
  - It sets a minimum bound on the time that device must wait before determining its parent has gone offline

- **Advantage**
  - It is accurate
Child Cache

![Percentage of Network Participating in Aggregate For Varying Amounts of Child Cache](image)

Redundancy

- Problem:
  - A node with multiple parents can send an aggregate value to both of instead of sending it to one
- Solution:
  - Linearly decompose the aggregate
  - Send part of the aggregate to one parent and rest to the other
  - Broadcast just a single message that is received and processed by both parents
  - This incurs no message overhead as long as both the parents
    - Are at the same level
    - Request data delivery during the same sub-interval of the epoch

Conclusion

- Declarative aggregate queries can be distributive and efficiently executed over sensor network
- The in-order approach provide an order of magnitude reduction in bandwidth consumption over centralized approach
- Query interface allows user to take advantage of aggregate operations without having to modify low-level code etc.
- Transparent optimizations decrease message cost and improve tolerance to failure and loss

Future Plans

- Extend this work as data collection needs of the wireless sensor community evolve
- Move towards an event-driven model where queries can be initiated and results can be collected in the response to the external event