Database Issues in Sensor Networks
Part II: In-Network Aggregation

Jun Yang
CPS 296.4, Fall 2007
SAMSI Course on Sensor Networks
Some slide contents come from J. Gao and A. Silberstein

A reminder on course projects
http://www.cs.duke.edu/courses/fall07/cps296.4/assignments.html
- Written project proposal
  - 1 page detailing group composition and topic
  - Due Oct. 16 (today!)
- Brief progress presentation
  - Approx. 5-10 min. for feedback
  - In Oct. 30 class
- Report
  - 10 page limit
  - Due Nov. 27 (last day of class)
- Conference-type presentation
  - 20-minutes + 5 min for questions in mini-conference
  - In Nov. 27 class

Review & roadmap
Database issues in sensor networks
- View sensor network as a distributed database
- Support declarative queries over the sensor network
- Part I: query processing using models
  - Model-driven pull
  - Model-based push for approximate data collection
- Part II: in-network data aggregation
  - Introduction: power of in-network processing
  - Robust aggregation
  - Continuous aggregation
An introduction to sensor aggregation


Computing aggregates

- SQL aggregates: MIN, MAX, SUM, COUNT, AVG
- More complex: COUNT(DISTINCT ...), median/quantiles, wavelets, samples, ...
- An aggregate function can be implemented with three functions:
  - Generate, $G(x)$: produce a partial state record from input
  - Fuse, $F(r_1, r_2)$: merge two records into one
  - Evaluate, $E(r)$: evaluate result from a partial state record
- E.g., for AVG:
  \[
  G(x) = (x, 1)
  \]
  \[
  F(hx, wy) = (x+y, w_x + w_y)
  \]
  \[
  E(hx, w_x) = x/w_x
  \]

Tree-based aggregation in TAG

- Each node generates
- Each internal node fuses records from its children with its own, and passes result record to parent
- Root node finally evaluates

*How does this beat collect-all-then-aggregate?
Making a tree more robust

- Tree is pretty fragile
  - If one link fails, data from entire subtree is lost
- Turn tree into a DAG?
  - Send 1/k of the summary to k parents, for free (broadcast)!
  - One link failure only drops 1/k data

Aggregation + routing spaghetti

- Variation of the DAG idea: send the whole summary up to k parents?
  - Works for some aggregates (which?)
  - But in general, one item can be counted many times!
- Aggregation scheme is too dependent on routing!
  - Routing tweaks affect correctness of aggregation
  - Can we decouple them?

Towards robust aggregation

Order and duplicate insensitivity (ODI)

Won’t it be nice if aggregation scheme is insensitive to the sequence or duplication of inputs?

- More precisely, a scheme is ODI-correct if, for any DAG, it produces a result identical to the correct answer produced by a canonical tree.

This is the property that made MIN/MAX easy.

Testing ODI-correctness

- Necessary and sufficient test turns out to be really simple:
  - $G(.)$ preserves duplicates; i.e., if $x_1$ and $x_2$ are considered duplicates, then $G(x_1) = G(x_2)$.
  - $F(., .)$ is commutative.
  - $F(., .)$ is associative.
  - $F(., .)$ is same-input idempotent; i.e., $F(r, r) = r$.

- Do MIN/MAX work?
- Does COUNT work out-of-box?

How to design ODI-correct schemes?

- Let’s do COUNT as an example.
  - A little randomness/approximation goes a long way.
  - Use synopses—compact, approximate summaries of data—for partial state records.
  - Borrow the “almighty” FM-sketch.
  - Then turn COUNT into MAX, which is ODI-correct.
FM-sketch

- Flajolet and Martin, 1985
- Counts # of distinct elements in a multi-set in one pass
  - Powerful building block for many data stream algorithms
- Start with a bitmap of 0’s: 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0
- For each element x in the multi-set, hash it to a positive integer using function h(x)
- Turn the h(x)-th bit on
  - # of distinct elements \( \approx \frac{2^{\text{position of first 0}} - 1}{0.77351} \)
- Use multiple independent h’s to improve accuracy
  - With enough number of h’s, can get within a prescribed error with probability higher than a prescribed threshold

FM-sketch (cont’d)

- For FM-sketch to work, need
  - \( \Pr[h(x) = 1] = 1/2, \Pr[h(x) = 2] = 1/4, \Pr[h(x) = 3] = 1/8, \ldots \)
  - Easy to simulate with a random binary hash \( g(x,i) \)
- Intuition: the i-th bit will be 1 if there are many more than \( 2^i \) distinct elements, each trying to set the bit with probability \( 1/2^i \)

Back to COUNT...

- Suppose each node has a unique id
- Partial state record: FM sketch with > \( \log n \) bits
- \( G(id) \): generate FM-sketch with \{id\}
- \( F(s_1, s_2) \): bitwise-OR the two input sketches
  - OR is like \( \text{MAX} \)
- \( E(s) \): estimate using the position of first 0 in \( s \)

  - How about \( \text{SUM} \)?
    - Convert to COUNT: for node id with integer value \( v \), generate \( v \) items \( \langle id, 1 \rangle, \langle id, 2 \rangle, \ldots, \langle id, v \rangle \)
Rings

- Now we can use much more flexible routing structures to help improve communication reliability without double-counting.

Snooping tricks

- Implicit acknowledgement
  - Explicit ack too expensive for sensor networks
  - Node $u$ sending to $v$ snoops subsequent transmissions from $v$ to see if $v$ indeed forwards the message for $u$
    - How does it work with synopses?
    - Why doesn’t this trick work for TAG SUM?
- Suppression
  - If my neighbor’s transmission subsumes mine, no need to transmit mine
    - Used in TAG
    - How would this trick work in synopsis diffusion?

Another example: uniform sample

- Suppose each node has a unique $id$
- $G(id, v) = \langle (id, v, r) \rangle$; $r$ is randomly chosen from $[0, 1]$
- Partial state record: a set of no more than $K$ entries of the form $(id, v, r)$
- $F(s_1, s_2)$: up to $K$ distinct entries in $s_1 \cup s_2$ with largest $r$
  - Again, top-$K$ is a simple extension of MAX
- $E()$: output all $(id, v)$ entries

- A random sample because $r$’s are randomly generated
Sensor aggregation problem: solved?

- How large are synopses?
- What are the costs of complex local processing?
- Is snooping completely free?
- MAX is not robust against outliers
  - What if somebody injects an all-1 FM-sketch?
- Everybody still transmits!
  - Can we do better?
- Are we taking advantage of spatio-temporal correlations?
  - Do suppression and redundancy really mix?

Towards continuous aggregation


Monitoring MAX

- Goal: for every epoch, return \( \langle \text{nodeid, value} \rangle \) for node with the highest value
  - A basic aggregation function useful in monitoring
    - How would TAG/synopsis diffusion do it?
    - What are the drawbacks?
- Suppression: don’t report if it doesn’t matter
  - Idea of this paper: leverage previous results as well
Adaptive range caching

- Root caches a range $R_i$ around each sensor reading $v_i$
- Reporting: sensor $i$ reports only if $v_i$ falls out of $R_i$; a new range is installed
  - Invariant: root knows $v_i \in R_i$
- Querying: sometimes root can determine the max node unambiguously from $R_i$'s
  - If not, query the nodes in question
- Adaptation: set ranges to balance
  reporting/querying costs
  - What trade-off can be controlled by adjusting range widths?
  - What do you think the ranges might look like eventually?

SLAT

Single-Level Adaptive Thresholds =
adaptive range caching with some tweaks

- Monitor the node with the current max $v_{\text{max}}$
- Ranges can be made one-sided = thresholds
  - Invariant: $v_i \leq t_i \leq v_{\text{max}}$
  - Report to root if value breaks threshold
  - If $v_{\text{max}}$ falls, query all nodes with $t_i > v_{\text{max}}$ to find new $v_{\text{max}}$ and set new thresholds

Example

- Root stores threshold

  Can we aggregate here?

  Unbroken threshold, no report

  Broken thresholds, all nodes report
Hierarchical adaptive thresholds

- Additional invariant: \( t_i \leq t_{i, \text{parent}} \)
- Subtree invariant: \( t_i \geq \text{all values in subtree} \)

- Each node remembers child thresholds
- Reporting: only if subtree invariant violated
- Querying: if max falls, only visit subtrees with threshold greater than the new candidate max
- Adjust thresholds in the process

Example
Suppression across space/time

- Thresholds at internal nodes carry over temporally
  - Example
    - Nodes a and b have common ancestors
    - Both rise, but b rises after a
    - b benefits from a’s earlier rise

= One node’s previous value help suppressing other nodes’ subsequent values
  - TAG only suppresses if a and b rise in the same epoch

Threshold setting: when?

- During reporting
  - Threshold may be forced to rise
    - When broken by any child
  - Or, by choice, threshold may rise higher than subtree max
    - If the node reported to its parent earlier in the epoch
      - Why would we do this?

- During querying
  - Threshold may be forced to fall
    - At or below new max
  - Or, by choice, threshold may be set lower than new max
    - If the node is contacted in querying
      - Why would we do this?

Threshold setting: where?

- Bisecting gap (guarding against an adversary)
- Using models that predict value trends
- Adaptively, similar to adaptive range caching
Discussion

- Too much reliance on the routing tree structure
- Suppressed reports vs. failures
- How can we apply the techniques in more general settings?
  - Other aggregates?
  - More complex queries: additional selections, group-bys, joins, etc.?
  - Multiple queries: what if MAX isn’t the only query running?
- What are the ideas that we can generalize?
  - Exploit query semantics
  - Exploit previous results
  - Exploit in-network processing and caching
  - Push vs. pull

Next time

- Alan Gelfand, Duke Statistics
- Stochastic modeling of data from spatio-temporal process
  - Slides already available on course Web site