Robust Aggregation in Sensor Networks

Jie Gao
Computer Science Department
Stony Brook University

Papers


Problem I: median

- Computing average is simple on an aggregation tree.
  - Each node x stores the average a(x) and the number of nodes in its subtree n(x).
  - The average of a node x can be computed from its children u, v. n(x)=n(u)+n(v). a(x)=(a(u)n(u)+a(v)n(v))/n(x).
- Computing the median with a fixed amount of message is hard.
  - We do not know the rank of u’s median in v’s dataset.
  - We resort to approximations.

Median and random sampling

- Problem: compute the median a of n unsorted elements {a_i}.
- Take a random sample of k elements. Compute the median x.
- Claim: x has rank within (1/2+ε)n and (1/2-ε)n with probability at least 1-2/exp{2kε^2}. (Proof left as an exercise.)
- Choose k=ln(2/δ)/2ε^2, then x is an approximate median with probability 1-δ.
- A deterministic algorithm?
- How about approximate histogram?
- What if a sensor generates a list of values?

Quantile digest (q-digest)

- A data structure that answers
  - Approximate quantile query: median, the kth largest reading.
  - Range queries: the kth to lth largest readings.
  - Most frequent items.
  - Histograms.
- Properties:
  - Deterministic algorithm.
  - Error-memory trade-off.
  - Confidence factor.
  - Support multiple queries.
Q-digest

- Exact data: frequency of data value \{f_1, f_2, \ldots, f_s\}.
- Compress the data:
  - Detailed information concerning frequent data are preserved;
  - Less frequently occurring values are lumped into larger buckets resulting in information loss.
- Buckets: the nodes in a binary partition of the range [1, s]. Each bucket v has range [v.min, v.max].
- Only store non-zero buckets.
- Digest property:
  - Count(v) ≥ n/k. (except leaf)
  - Count(v) + Count(p) + Count(s) > n/k. (except root)

Example

Input data bucketed

Q-digest

Construct a q-digest

- Each sensor constructs a q-digest based on its value.
- Check the digest property bottom up: two “small” children’s count are added up and moved to the parent.

Merging two q-digests

- Merge q-digests from two children
- Add up the values in buckets
- Re-evaluate the digest property bottom up.

Information loss: t undercounts since some of its value appears on ancestors.

Space complexity and error bound

1. A q-digest with compression parameter k has at most 3k buckets.
- By property 2, for buckets Q,
  - \( \sum_{v \in Q} [\text{Count}(v) + \text{Count}(p) + \text{Count}(s)] > |Q| \cdot n/k. \)
  - \( \sum_{v \in Q} [\text{Count}(v) + \text{Count}(p) + \text{Count}(s)] \leq 3\sum_{v \in Q} \text{Count}(v) = 3n. \)
  - \(|Q| < 3k. \)
- Any value that should be counted in v can be present in one of the ancestors.
  - Count(v) has max error \log n/k.
  - Error(v) \leq \sum_{\text{ancestors}} \text{Count}(p) \leq \sum_{\text{ancestors}} n/k \leq \log n/k.
  - MERGE maintains the same relative error.
  - Error(v) \leq \log n/k \leq \log n/k.

Median and quantile query

- Given qc(0, 1), find the value whose rank is qn.
- Relative error \( e = |r - qn|/n, \) where r is the true rank.
- Post-order traversal on Q, sum the counts of all nodes visited before a node v, which is the lower bound on the # of values less than v.max.
  - Report it when it is first time larger than qn.
- Error bound: \( \log e/k = \log e/m, \) where m = 3k is the storage bound for each sensor.
Other queries

- **Inverse quantile**: given a value, determine its rank.
  - Traverse the tree in post-order, report the sum of counts $v$ for which $x>v$.max, which is within $[\text{rank}(x), \text{rank}(x)+e]\$.

- **Range query**: find # values in range $[l, h]$.
  - Perform two inverse quantile queries and take the difference. Error bound is $2e$.

- **Frequent items**: given $s \in (0, 1)$, find all values reported by more than $sn$ sensors.
  - Count the leaf buckets whose counts are more than $sn$.
  - Small false positive: values with count between $(s-e)n$ and $sn$ may also be reported as frequent.

Simulation setup

- A typical aggregation tree (BFS tree) on 40 nodes in a 200 by 200 area. In the simulation they use 4000–8000 nodes.

Simulation setup

- Random data;
- Correlated data: 3D elevation value from Death Valley.

Histogram v.s. q-digest

- Comparison of histogram and q-digest.

Tradeoff between error and msg size

- Comparison of histogram and q-digest.

Saving on message size

- Comparison of histogram and q-digest.
Problem II: Aggregation along a spanning tree in practice

• The impact of link dynamics on aggregation tree.
  • If a link fails, the data from the entire subtree is lost.
    – Wrong aggregated value;
    – Inconsistency.

• Solution: use multi-path routing (e.g., DAG) to improve robustness under link dynamics.
  • But if both paths succeed, the same data is received twice!
  • This is ok for some aggregation such as MAX, MIN.
  • How about Count, SUM?

Aggregation along a spanning tree

• Problem with spanning tree: Link dynamics
  • If a link fails, the data from the entire subtree is lost.
  • Decouple routing and data aggregation.
  • Use multi-path routing to improve the routing robustness.
  • If multiple paths succeed, the sink receives multiple copies of the same data.
  • Design an aggregation algorithm that is insensitive to order and duplications.

Order and duplicate insensitive (ODI) synopses

• Aggregated value is insensitive to the sequence or duplication of input data.
  • Small-sizes digests such that any particular sensor reading is accounted for only once.
    – MAX, MIN admit natural ODI synopsis.
    – ODI synopsis for SUM, COUNT, MEDIAN, AVG are more challenging.
  • Synopsis generation: SG().
  • Synopsis fusion: SF() takes two synopsis and generate a new synopsis of the union of input data.
  • Synopsis evaluation: SE() translates the synopsis to the final answer.

ODI synopsis for MAX/MIN

• Synopsis generation: SG().
  – Output the value itself.
• Synopsis fusion: SF() (x)
  – Take the MAX/MIN of the two input values.
• Synopsis evaluation: SE().
  – Output the synopsis.

ODI correctness

• A synopsis diffusion algorithm is ODI-correct if SF() and SG() are order and duplicate insensitive functions.
  • Or, if for any aggregation DAG, the resulting synopsis is identical to the synopsis produced by the canonical left-deep tree.
  • The final result is independent of the underlying routing topology.
ODI-synopsis

Connection to streaming model: data item comes 1 by 1.

(a) Aggregation DAG
(b) Canonical left-deep tree

Test for ODI correctness

1. \( \text{SG}() \) preserves duplicates: if two readings are considered duplicates (e.g., two nodes with the same temperature readings), then the same synopsis is generated.
2. \( \text{SF}() \) is commutative.
3. \( \text{SF}() \) is associative.
4. \( \text{SF}() \) is same-synopsis idempotent, \( \text{SF}(s, s) = s \).

Theorem: The above properties are sufficient and necessary properties for ODI-correctness.

Proof idea: transfer an aggregation DAG to a left-deep tree with the same output by using these properties.

Proof of ODI correctness

1. Start from the DAG. Duplicate a node with out-degree \( k \) to \( k \) nodes, each with out degree 1. \( \text{SG}() \) duplicates preserving.

2. Re-order the leaf nodes by the increasing value of the synopsis. \( \text{Commutative} \).

3. Re-organize the tree so that adjacent leaves with the same value are input to a SF function. \( \text{Associative} \).

4. Replace \( \text{SF}(s, s) \) by \( s \). \( \text{Same-synopsis idempotent} \).
Proof of ODI correctness

5. Re-order the leaf nodes by the increasing canonical order. \(\text{Commutative.}\)

Counting distinct elements

- Each sensor generates a sensor reading. Count the total number of different readings.
- Coin tossing experiments \(CT(x) = \# \text{ coin tosses until the first head occurs or } x \text{ coin tosses with no heads.}\)
- \(Pr[CT(x)=i] = Pr[i-1 \text{ tails and 1 head}] = 2^{-i}.\)
- Use a pseudo-random generator so that \(CT(x)\) is a hash function.

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

Counting distinct elements

- Synopsis: a bit vector of length \(k > \log n.\)
- \(SG(i):\) output a bit vector \(s\) of length \(k\) with \(CT(k)\)'s bit set.
- \(SF(i):\) bit-wise boolean OR of input \(s\) and \(s'.\)
- \(SE(i):\) if \(s\) is the lowest index that is still \(0,\) output \(2^{-i}/0.77351.\)

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 0 \\
\text{OR} \\
0 & 1 & 0 & 0 & 0 & 0 \\
\downarrow \\
i=3
\end{array}
\]

Distinct value counter analysis

- Lemma: For \(i < \log n - 2\log \log n,\) \(FM[i] = 1\) with high probability (asymptotically close to 1). For \(i \geq 3/2 \log n + \delta,\) with \(\delta \geq 0,\) \(FM[i] = 0\) with high probability.
- The expected value of the first zero is \(\log(0.7753n) + P(\log n) + o(1),\) where \(P(u)\) is a periodic function of \(u\) with period 1 and amplitude bounded by \(10^{-5}.\)
- The error bound (depending on variance) can be improved by using multiple copies or stochastic averaging.

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 0 \\
\text{OR} \\
0 & 1 & 0 & 0 & 0 & 0 \\
\downarrow \\
i=3
\end{array}
\]

Sum

- Naïve approach: for an item \(x\) with value \(c\) times, make \(c\) distinct copies \((x, j), j = 1, \ldots, c.\) Now use the distinct count algorithm.
- When \(c\) is large, we set the bits as if we had performed \(c\) successive insertions to the FM sketch.
- First set the first \(\delta = \log \log n\) bits to 1.
- Those who reached \(\delta\) follow a binomial distribution: each item reaches \(\delta\) with prob \(2^{-k}.\)
- Explicitly insert those that reached bit \(\delta\) by coin flipping.
- Powerful building block.
Second moment

- Kth moment $\mu_k = \sum x^k$, $x_i$ is the number of sensor readings with value $i$.
  - $\mu_0$ is the number of distinct elements.
  - $\mu_1$ is the sum.
  - $\mu_2$ is the square of $L_2$ norm.
- The famous sketch algorithm for frequency moments can be turned into an ODI easily by using ODI-sum.

The space complexity of approximating the frequency moments, N. Alon, Y. Matias, and M. Szegedy. STOC 1996.

Uniform sample

- Each sensor has a reading. Compute a uniform sample of a given size $k$.
- Synopsis: a sample of $k$ tuples.
- SG(): output (value, $r$, id), where $r$ is a uniform random number in range $[0, 1]$.
- SF(): output the $k$ tuples with the $k$ largest $r$ values. If there are less than $k$ tuples in total, output them all.
- SE(): output the values in s.
- ODI-correctness is implied by the “MAX” and union operation in SF().
- Correctness: the largest $k$ random numbers is a uniform $k$ sample.

Most popular items

- Return the $k$ values that occur the most frequently among all the sensor readings.
- Synopsis: a set of $k$ most popular items.
- SG(): output (value, weight) pair, with weight=$CT(k)$, $k \geq \log n$.
- SF(): for each distinct value $v$, discard all but the pair with max weight. Then output the $k$ pairs with max weight.
- SE(): output the set of values.
- Note: we attach a weight to estimate the frequency.
- Many aggregates that can be approximated by using random samples now have ODI-synopsis, e.g., median.

Implicit acknowledgement

- Explicit acknowledgement:
  - 3-way handshake.
  - Used on internet.
- Implicit acknowledgement:
  - Used on ad hoc wireless networks.
  - Node $u$ sending to $v$ snoops the subsequent broadcast from $v$ to see if $v$ indeed forwards the message for $u$.
  - Explores broadcast property, saves energy.
- With aggregation this is problematic.
  - Say $u$ sends value $x$ to $v$, and subsequently hears value $z$.
  - $U$ does not know whether or not $x$ is incorporated into $z$.

Implicit acknowledgement

- ODI-synopsis enables efficient implicit acknowledgement.
  - $U$ sends to $v$ synopsis $x$.
  - Afterwards $u$ hears that $v$ transmitting synopsis $z$.
  - $U$ verifies whether SF($x$, $z$)=$z$?
Decouple routing with aggregation

- Use arbitrary multi-path routing schemes to allow message redundancy to be adaptive to sensor network conditions.
- Use Directed Acyclic Graph (DAG) to replace tree.
- Rings overlay:
  - Query distribution: nodes in ring $R_j$ are $j$ hops from $q$.
  - Query aggregation: node in ring $R_j$ wakes up in its allocated time slot and receives message from nodes in $R_{j+1}$.

Rings and adaptive rings

- Adaptive rings: cope with network dynamics, node deletions and insertions, etc.
- Each node on ring $j$ monitor the success rate of its parents on ring $j-1$.
- If the success rate is low, the node connects to other node whose transmission is overhead a lot.
- Nodes at ring 1 may transmit multiple times to ensure robustness.

Error of approximate answers

- Two sources of errors:
  - Algorithmic error: due to randomization and approximation.
  - Communication error: the fraction of sensor readings not accounted for in the final answer.
- Algorithmic error depends on the choice of algorithm and thus relatively controllable.
- Communication error depends on the network dynamics and robustness of routing algorithms.

Simulation results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>% nodes</th>
<th>Error (Uniform)</th>
<th>Error (Skewed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG</td>
<td>21%</td>
<td>0.85</td>
<td>0.99</td>
</tr>
<tr>
<td>RINGS</td>
<td>45%</td>
<td>0.33</td>
<td>0.19</td>
</tr>
<tr>
<td>ADAPT, RINGS</td>
<td>100%</td>
<td>0.15</td>
<td>0.16</td>
</tr>
<tr>
<td>FLOOD</td>
<td>100%</td>
<td>0.15</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Conclusion

- Due to the high dynamics in sensor networks, robust aggregates that are insensitive to order and duplication are very attractive – they provide the flexibility of using any multi-path routing algorithms and re-transmission.
- Use ODI-synopsis as black box operators to replace naïve operators in more complex data structures.