More ODI synopsis

• Distinct values
• SUM
• Second moment
• Uniform sample
• Most popular items
• Set membership --- Bloom Filter
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 “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>logn.
- 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.
Set membership: Bloom Filter

• A compact data structure to encode set containment.
• Widely used in networking applications.

• Given: n elements $S=\{x_1, x_2, \ldots, x_n\}$.
• Answer query: whether $x$ is in $S$?

• Allow a small false positive (an element not in $S$ might be reported as “yes”).
Bloom filter

• An array of \( m \) bits.
• Insert: for \( x \in S \), use \( k \) random hash functions and set \( h_j(x) \) to “1”.
• Query: to check if \( y \) is in \( S \), search all buckets \( h_j(y) \), if all “1”, answer “yes”.
• No false negative. Small false positive.
Bloom filter tricks

• Union of $S_1$ and $S_2$:
  – Take “OR” of their bloom filters.
  – ODI aggregation.

• Shrink the size to half:
  – OR the first and second halves.
Counting bloom filter

- Handle element insertion and deletion
- Each bucket is a counter.
- Insert: increase by “1” on the hashed locations.
- Delete: decrease by “1”.

- Be careful about buffer overflow.
Spectral bloom filter

- Record multi-set \( \{x_1, x_2, \ldots, x_n\} \), each item \( x_i \) has a frequency \( f_i \).
- Insert: add \( f_i \) to each bucket.
- Retrieve: return the smallest bucket value from the hashed locations.
- Idea: the smallest bucket is unlikely to be polluted.
Bloom filter applications

• Traditional applications:
  – Dictionary, UNIX-spell checker.

• Network applications:
  – Cache summary in content delivery network.
  – Resource routing, etc.
  – Read the survey for more….

• Good for sensor network setting:
  – ODI, compact, many algebraic properties.
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.
Is the problem solved? NO

• Best effort multi-path routing does not guarantee all data have been incorporated.
  – Blackbox setting.

• ODI synopsis translates everything to MAX, which is not robust to outliers!
  – Sensor malfunction.
  – Malicious attacks.

• For exemplary aggregations (MAX, MIN), the final result is a single sensor value, but all nodes are examined. – Can we improve?
CountTorrent

- To improve routing robustness, deliver each value multiple times to make sure at least one copy arrives
  - Synopsis diffusion: aggregation of the same value for multiple times does **not** result in double counting.
  - CountTorrent: remember what value has been included in the aggregation in an implicit manner.
How to record the members in the aggregate?

• In the naive way, keep the members explicitly.
  – Storage cost /communication cost too high
  – It loses the point of aggregation.
• In the implicit way
  – Label the aggregate
CountTorrent

- Each node has a label: a 0,1 string
- Two nodes can have their data aggregated if their labels are the same except the last bit.
- After aggregation, remove the last bit and assign the label to the aggregated data.
- Gossip-style communication: each node exchanges its value with neighbors.
CountTorrent example

- For any 2 nodes, their labels are neither the same nor is either one a substring of the other.
- All $N$ labels can be merged pairwise and recursively to yield $\epsilon$, the empty string.
Aggregation

- Each node keeps a buffer of received value/label pair
- Consolidate: try to merge the data in the buffer

```
Node i's prefix buffer          new tuple
Step 1:  (011, 3)  (101, 2)  (0101, 1)  +  (0100, 3)

Step 2:  (011, 3)  (101, 2)  (010, 4)

Step 3:  (01, 7)  (101, 2)
```
How to assign labels?

- Each node is given the label of a leaf node.
Conclusion

• Aggregation sometimes requires careful design to tradeoff accuracy & storage/message size.

• Aggregation incurs information loss, making robust estimation more difficult. E.g. a single outlier reading can screw up MAX/MIN aggregates.
Multi-dimensional Data and Spatial Range Query in Sensor Networks
Papers


Orthogonal range search

- Find all the sensors inside a rectangular box.
- Find all the sensors with temperature readings above 70F.
Multi-dimensional data

- Monitor environments.
- Multiple sensors, multiple attributes.
- Query might be multi-dimensional as well.

List all sensors with temperature value 70-80 and light level 10-20.
Sensor network as a database

- Need an indexing scheme.
- ... In addition, a storage scheme.

- First we look at range query in a centralized setting.
1D range search

- Find the data inside a query interval \([x, x']\)
- 1D range tree: a balanced partitioning tree on a sorted list.
  - Each leaf stores an input value.
  - Each internal node stores the splitting value.
1D range search

- Find the data inside a query interval \([x, x']\)
  - Start from the root and descend the tree to find the interval where \(x\) and \(x'\) stays.
  - Include all the leaves in the sub-trees between the two traversing paths from the root.

- Example \([9, 33]\).
1D range search

- Storage: $n + n/2 + n/4 + \ldots + 1 = 2n = O(n)$
- Height of the tree: $O(\log n)$
- Query time: $O(\log n + k)$, where $k$ is the output size.