CS7960 L26 : distrib | Mergeable Summaries

distributed nodes

Many nodes in graph
 - each node knows only small number of neighbors
 - need to communicate of calculate

key bottleneck is communication

Mergeable Summaries:

Many unorganized nodes [1,...,k] each with data X_i.
 <Connected in tree structure>

X = cup_i X_i

Want S = summ(X), but don't want to send X.

Key operation:
  - given S_1 = summ(X_1) and S_2 = summ(X_2)
  - produce S_12 = summ(X_1 cup X_2)

Example: X_1 = {1,2,3,8,9}
          X_2 = {4,5,89,90,91}
          X_3 = {6,7,92,93,94}
m1 = median(X_1) = 3
m2 = median(X_2) = 89
m3 = median(X_3) = 92
median{m1,m2,m3} = 89
median(X1 cup X_2 cup X_3) = 8

often error (or size) accumulates
----

goal: S = summ(X) is a eps-approximation of X

X multi-subset [n]
f_i = |{x_j in X | x_j = i}|

eps-approx frequency values
 |~f_i - f_i| <= eps F_1 = eps m

```
size S = 1/eps
-----

- error is relative
- size depends only on eps

key operation:
 given:   S_1 = summ(X_1), S_2 = summ(X_2)
       - S_i is eps-approx of X_i
       - size(S_i) = f(1/eps)
 output:  S_12 = summ(X_1 cup X_2)
       - S_12 is eps-approx of X_1 cup X_2
       - size(S_12) = f(1/eps)

 * neither size, nor error increase

-----------------

Misra-Gries Summaries:
S =
Let C be array of k counters C[1], C[2], ..., C[k]
Let L be array of k locations L[1], L[2], ..., L[k]

S_1 = (C_1, L_1) = summ(X_1)
S_2 = (C_2, L_2) = summ(X_2)

k = 1/eps = 3

S_12  [1 + 0] [2 + 3] [0 + 4] [0 + 0] [3 + 0] [0 + 2]
   -> [1]     [5]     [4]     [0]     [3]     [2]*
   -> [0]     [3]     [2]     [0]     [1]     [0]

 - add like counters together (at most 2k)
 - retain just top k after subtracting C[k+1], set rest to 0.

proof:
 Each subtraction removes >= k items
 can subtract at most m/k times
 each value ~f_i in [f_i, f_i - m/k] = [f_i, f_i - eps m]

-----------------

commutative, associative

Any linear summary:
  sum(X_12) = sum(X_1) + sum(X_2)
```

```
Any idempotent summary:
  max(X_12) = max{max(X_1), max(X_2)}

-----------------

count-min sketch

*****************************
t independent hash functions {h_1, ..., h_t}
each h_i : [n] -> [k]

2-d array of counters:
h_1 -> [C_{1,1}] [C_{1,2}] ... [C_{1,k}]
h_2 -> [C_{2,1}] [C_{2,2}] ... [C_{2,k}]
...    ...                      ...
h_t -> [C_{t,1}] [C_{t,2}] ... [C_{t,k}]

for each a \in A -> increment C_{i,h_i(a)} for i in [t].

hat{f}_a = min_{i in [t]} C_{i,h_i(a)}

Set t = log(1/delta)
Set k = 2/eps
*****************************

can add or subtract!

----------------

eps-RELATIVE-RANK:
Build data structure S.
rank(v) = 1 + # items in A smaller than v
relative-rank(v) = Rrank(v) = rank(v)/|X|  in [0,1]

eps-RELATIVE-RANK S returns S(v) such that
 Rrank(v) - eps <= S(v) <= Rrank(v) + eps


11111111111
Random Sample size k = O(1/eps^2) = S
Rrank_S(v) = S(v)
| Rrank(v) - S(v) | <= eps

S_1 = {(s_1, u_1), (s_2, u_2), ...}
S_2 = {(s_1, u_1), (s_2, u_2), ...}
  - u_i at random for each s_i
  - keep top k values u_i (and paired s_i)
```

easily mergeable, maintain random sample size k.

22222222222
Maintain sorted list of size k = O(1/eps sqrt(log(1/eps)))
S_1 = {s_{11}, s_{12}, s_{13}, ..., s_{1k}}
S_2 = {s_{21}, s_{22}, s_{23}, ..., s_{2k}}
s.t. s_{i,j} < s_{i,j+1} for i = {1,2}

S_12 =
1. merge sort S_1, S_2 -> ordered list size 2k
2. select even points / odd points at random

***magically, error does not accumulate, nor probability of failure
older merges less important towards relative error

above only works for |X_1| = |X_2|

if not true, need size O((1/eps) (log(1/eps))^3/2)