MCMD L19 : MapReduce | Sorting + Sliding Windows

MapReduce

S = Massive Data

Mapper(S): s in S -> {(key,value)}

Shuffle({(key,value)}) -> group by "key"

Reducer ({"key,value_i}) -> ("key, f(value_i))

Can repeat, constant # of rounds

______

[Tao + Lin + Xiao 2013]

Minimal MapReduce Algorithm
  N = size of data
  t = number of machines
  m = N/t = # objects per machine if distributed evenly.
      m < M = Mem size

  1) At all times each machine has O(m) storage
  2) Each machine sends/receives O(m) items
  3) constant # rounds
  4) Optimal computation: each machine performs O(T_seq / t) in total
    O(T_seq / t) per machine per round.

  1)+2) prevents partition skew
     m = N/t allows to scale to any # machines !
  2) ensures total traffic is O(N)
     no straggling machine
     ensures is stateless (resilience, can use fake larger t + load balancing)
  3) for practicality
  4) energy cost is low

Sorting

TeraSort: http://sortbenchmark.org
  Ellapsed time to sort 10^12 bytes = 1TB   (now 100 TB)
  measured in TBs/minute
    -> record (2013) 1.42 TB minute on 102.5 TB.

2009: Hadoop 100 TB in 172 min (0.572 TB / min) (3452 machines)

```
      500 GB in 1 minute (on 1406 machines)
      previous used fewer, but expensive machines


How does it work?

   parameter k = t ln (N*t)
---------------------------
Map 1:
  For all s in S, with prob (k/N)
     -> {<1,s> <2,s> ... <t,s>}   <original TeraSort, only send to 1>

Reduce 1:
  On each node:  <j, {s_1 ... s_{~k}} = Q>   (same Q)
    -> sort(Q), choose t-1 even spaced items b_1, b_2, ..., b_{t-1}
       b_j = j[k/t]th item
       b_0 = -infinity, b_t = infinity


Map 2:
  For all s in S:  find j s.t. b_{j-1} < s <= b_j
    -> <j, s>

Reduce 2: <j, {s, s', …} = S_j}
  -> <j, sort(Q_j)>
---------------------------

Central Limit theorem (Chernoff Bound)  k/2 < |Q| < k  w.h.p.

Need:
 (1) |Q| = O(m)    fine for t = O(m / log (N))
 (2) for all j,  |S_j| = O(m)

Given (2), then T_j = (N/t) log (N/t)
           sum_j T_j = (N/t) log (N/t) = N log (N/t) < N log N

Prove (2):
  eps-net:  Given k = (1/eps) ln (1/eps * delta) samples, w.p > 1-
delta:
               each interval of size eps*N has at least one point
               -> each |S_j| <= N/t + 2*eps*N
                  (not completely obvious, symmetric difference)
          set eps*N = N/t ->  t = 1/eps
               -> k = t ln (t/delta)
          w.p.h = w.p > 1-1/N ->  k = t ln (tN)


-----------
Makes many tasks Minimal:  e.g. Prefix Sum:
  Sort (2 rounds)
```

```
Reduce2: also computes agg(S_j) = sum(S_j) = g_j
   -> {<1,g_j> <2,g_j> … <j,g_j>}
   -> {<j,s>  for all s in S_j}

Map3: identity

Reduce3: node j:
          W_j = sum_{i=1}^j g_j
          for s_i in S
             W_j += w_i
             p_i = W_j



-----------------------------------------
Sliding Aggregates

S has N objects:  ordered, each s_i has weight w_i
integer l < N
distributed aggregate agg (e.g. Sum, Min, Max)
   for S1 and S2  have agg(agg(S1), agg(S2)) = agg(S1 union S2)
window(i) = l largest items not exceeding s_i

sliding window statistics


Rounds 1+2 -->  Sort ->  S_1, ..., S_t

Round 3  -> use rank (prefix sum w/ w_i = 1) to have each |S_j| = m
*exacty*


Round 4:
Map 4:  (really Reduce 3)
  + Send A_j = agg(S_j) to all machines
      {<1,A_j> , <2,A_j>, ..., <t,A_j>
  + Send <[(i-l)/t],w_i>  for all s_i in S_j

Reduce:
  window(i) = agg(
              agg_{l = i-l}^[(i-l+1)/t]*k w_i
              , A_[(i-l+1)/t] , A_[(i-l+2]/t , ... , A_[(i-1)/t]
              , agg_{s_l in S_j, l<i} w_i
    can be done in O(m) time

** each s_i important for at most 2 units, we know which ones **
```