L24 -- Efficient Page Rank
[Jeff Phillips - Utah - Data Mining]

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

MapReduce

Big data D = \{D_1, D_2, ... D_m\}
too big for one machine
each D_i on machine i

[ Each machine has limited memory! ... compared to data ]

proceeds in rounds (3 parts):
1: Mapper
   all d in D \rightarrow (k(d), v(d))
2: Shuffle
   moves all (k, v) and (k', v') with k=k' to same machine
3: Reducer
   \{(k,v_1), (k,v_2), ...,\} \rightarrow output usually f(v_1,v_2,...)

1.5: Combiner
   if one machine has multiple (k,v_1), (k,v_2)
   then performs part of Reduce before Shuffle.

Can think of output of Reducer as D_i on machine i.
Then can string multiple MR-rounds together.

*** key-value pairs can encode much deeper computing power
   + Mapper f(D_i) \rightarrow \{(k_i,v_i)\}_j \rightarrow with (k_i = i, v_i = input to node i)
*** Provides very robust system, many fail-safes if node goes down, gets slow...
*** very simple!

-------- EXAMPLE --------

Histogram into k bins
Mapper d in D \rightarrow (k=bin(d), 1)
   (combiner)
Reducer (k=i,v) \rightarrow output = sum v

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

Page Rank:

Internet stored as big matrix M (size=nxn)
   + sparse, 99%+ of entries are 0
   ([M[a,b] = 0] == no link from page a to page b)
+ P = beta M + (1-beta) B where B[a,b] = 1/n
beta \approx 0.85

page-rank vector: q_* = P^t q as t-> infty (here t = 50 to 75 ok)
"importance of webpage" (other details too, but this is computational hard part)

Problems:
- M is sparse, but B (implicit) and P^n is dense! Too BIG to store
  -- q_i is O(n) can always store, so just compute
    q_{i+1} = beta * M * q_i + (1-beta) e/n
times
- Still very big computation. Gigabytes.
  Many machines and machine crash!
  -- MapReduce!

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

simple: assume q fits in one machine (twice: e.g. q_i and q_{i+1})

-- break M into vertical stripes
M = [M1 M2 ... Mk]
(and q into q = [q1; q2; ...; qk] = horizontal split)
then
Mapper i \rightarrow (key=i' in [k]; val = (row=r of Mi * q_i))
Reducer: adds values to get each element q[i'] * beta + (1-beta)/n

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

big q: what if q does not fit in a single machine?

option 1: Tiling.

M into sqrt(k) x sqrt(k) blocks
M = [M11 M12 .. M1sqrt{k};
    M21 M22 .. M2sqrt{k};
    ...
    Msqrt{k}1  Msqrt{k}2 .. Msqrt{k}sqrt{k}]

Mapper:
k machines each get one block M_{i,j}
and get sent q_i for i in [sqrt{k}]

Reducer:
on each row i', adds M_{i,j} q_i -> q[i']
and does $q_+[i'] = q[i'] \* \beta + (1-\beta)/n$

Problems:
- each $q_i$ (for $i$ in $[\sqrt{k}]$) is sent $\sqrt{k}$ places
- thrashing: on $M_{i,j}$
  --> solution: striping -> prefetching
    on $q_+$ (each column $M_{i,j}$ may add to $q_+[i']$)
  --> solution: blocking on $M_{i,j}$ ($\sqrt{k} \times \sqrt{k}$ blocks)
    read $M_{i,j}$ once || read,write $q/q_+$ $\sqrt{k}$ times

Example:

\[
M = \begin{bmatrix}
0 & 1/2 & 0 & 0 \\
1/3 & 0 & 1 & 1/2 \\
1/3 & 0 & 0 & 1/2 \\
1/3 & 1/2 & 0 & 0
\end{bmatrix}
\]

stripe:
- $M_1 = \begin{bmatrix}
0; 1/3; 1/3; 1/3
\end{bmatrix}$
  stored as (1: (1/3,2) (1/3,3) (1/3,4))
- $M_2 = \begin{bmatrix}
1/2; 0; 0; 1/2
\end{bmatrix}$
  stored as (2: (1/2,1) (1/2,4))
- $M_3 = \begin{bmatrix}
0; 1; 0; 0
\end{bmatrix}$
  stored as (3: (1,3))
- $M_4 = \begin{bmatrix}
1/3; 1/2; 0; 0
\end{bmatrix}$
  stored as (4: (1/3,1) (1/2,2))

block:
- $M_{11} = \begin{bmatrix}
0 1/2; 1/3 0
\end{bmatrix}$
  stored as (1: (1/2,2)) (2: (1/3,1))
- $M_{12} = \begin{bmatrix}
0 0; 1 1/2
\end{bmatrix}$
  stored as (4: (1,1) (1/2,2))
- $M_{21} = \begin{bmatrix}
1/3 0; 1/3 1/2
\end{bmatrix}$
  stored as (1: (1/3,3)) (2: (1/3,3) (1/2,4))
- $M_{22} = \begin{bmatrix}
0 1/2; 0 0
\end{bmatrix}$
  stored as (3: (1/2,4))

Note that some blocks have no effect on some vector elements they are responsible for
  --> $M_{22}$ has no effect on $q_+[3]$.
  --> $M_{12}$ has no use for $q[3]$.
  This is quite common, and can be used to speed up.