MapReduce

Big data $D = \{D_1, D_2, \ldots, D_m\}$
 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 $D$  -> $(k(d), v(d))$
2. Shuffle
   moves all $(k, v)$ and $(k', v')$ with $k=k'$ to same machine
3. Reducer
   $\{(k,v_1), (k,v_2), \ldots,\}$  -> output usually $f(v_1,v_2,\ldots)$

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 $f(D_i)$  -> $\{(k_i,v_i)\}_j$  -> with $(k_i = i, v_i = \text{input to node } i)$
*** Provides very rubout system, many fail-safes if node goes down, gets slow...
*** very simple!

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

Histogram into k bins
Mapper $d$ in $D$  -> $(k=\text{bin}(d), 1)$
   (combiner)
Reducer  $(k=i, v) -- \text{output} = \text{sum } v$

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

Page Rank:

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

Page-rank vector: \( q_* = P^t q \) as \( t \to \infty \) (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
    \[
    q_{[i+1]} = \beta * M * q_i + (1-\beta) e/n
    \]
    \( t \) 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
      \[
      M = [M1 \ M2 \ldots \ Mk]
      \]
      (and \( q \) into \( q = [q_1; q_2; \ldots; q_k] \) = horizontal split)
      then
      Mapper \( i \to (key=i' \text{ in } [k] \); val = (row=r of \( M_i \) * \( q_i \)) \)
      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
\[
M = [M_{11} \ M_{12} \ldots M_{1\sqrt{k}}; \ M_{21} \ M_{22} \ldots M_{2\sqrt{k}}; \ldots; \ M_{\sqrt{k}1} \ M_{\sqrt{k}2} \ldots M_{\sqrt{k}\sqrt{k}}]
\]

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 \( M_\{i,j\} \) \( q_i \) to \( q[i'] \)
and does \( q_+[i'] = q[i'] \ast \beta + (1-\beta)/n \)

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:
\[
M1 = \begin{bmatrix}
0; 1/3; 1/3; 1/3 \\
\end{bmatrix}
\]
stored as \( (1: (1/3,2) (1/3,3) (1/3,4)) \)
\[
M2 = \begin{bmatrix}
1/2; 0; 0; 1/2 \\
\end{bmatrix}
\]
stored as \( (2: (1/2,1) (1/2,4)) \)
\[
M3 = \begin{bmatrix}
0; 1; 0; 0 \\
\end{bmatrix}
\]
stored as \( (3: (1,3)) \)
\[
M4 = \begin{bmatrix}
1/3; 1/2; 0 0 \\
\end{bmatrix}
\]
stored as \( (4: (1/3,1) (1/2,2)) \)

block:
\[
M11 = \begin{bmatrix}
0 1/2; 1/3 0 \\
\end{bmatrix}
\]
stored as \( (1: (1/2,2)) (2: (1/3,1)) \)
\[
M12 = \begin{bmatrix}
0 0; 1 1/2 \\
\end{bmatrix}
\]
stored as \( (4: (1,1) (1/2,2)) \)
\[
M21 = \begin{bmatrix}
1/3 0; 1/3 1/2 \\
\end{bmatrix}
\]
stored as \( (1: (1/3,3)) (2: (1/3,3) (1/2,4)) \)
\[
M22 = \begin{bmatrix}
0 1/2; 0 0 \\
\end{bmatrix}
\]
stored as \( (3: (1/2,4)) \)

Note that some blocks have no effect on some vector elements they are responsible for
  --> \( M22 \) has no effect on \( q_+[3] \).
  --> \( M12 \) has no use for \( q[3] \).
  This is quite common, and can be used to speed up.