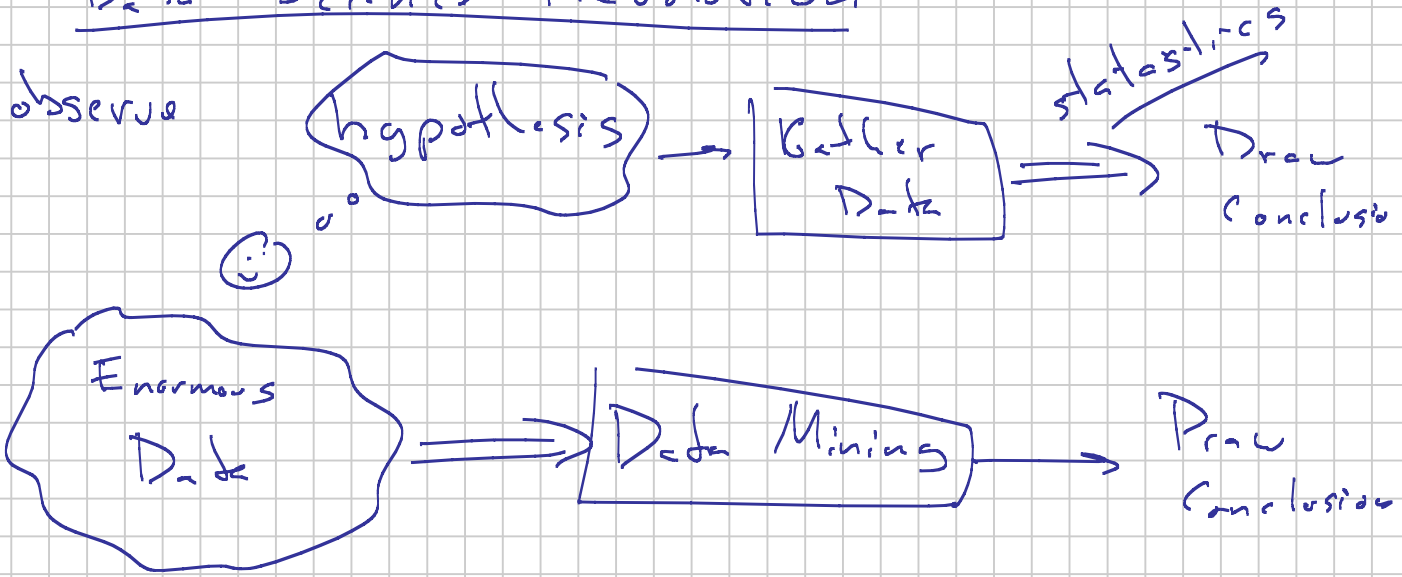
MapReduce + DFS

Note Title 4/13/2016

Big Data

- Data in Enormous ... ≥ Terabytes
  - ↳ 100 or 1000s machines
- Data static. Usually appends.

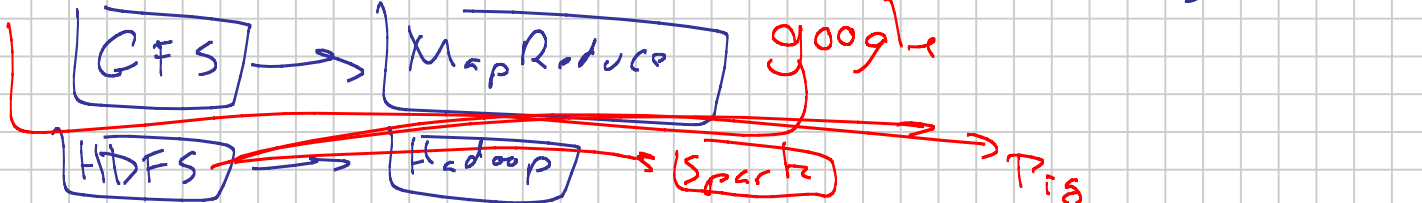
Data Science Revolution

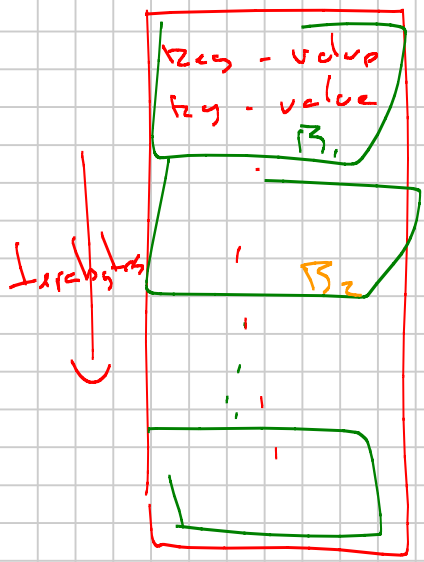


Distributed File Systems

key-value pairs

key	value
log id	log
web address	html, outgoing links
doc id	list of words
word	list of documents containing word

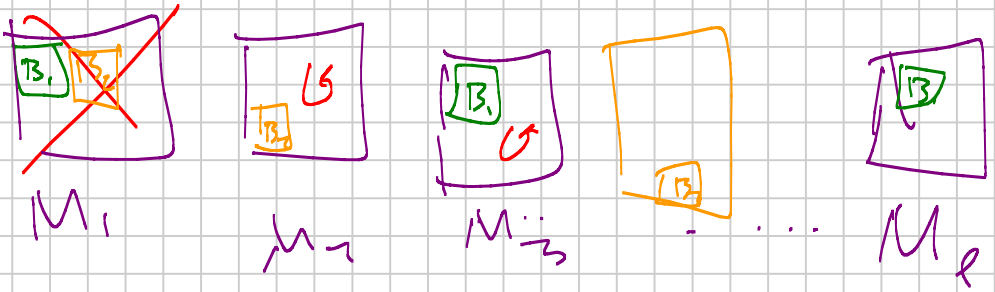




blocks Blocks = {B1, B2, ... Bn}

64 MB Bi x 2 or 3 copies

no locality (random)



- Redundancy (Disk failure) → Resilancy
- Heterogeneity

Map Reduce

1. Map — operates on data in DFS
   map(<k1, v>) → {<k1', v1'>, <k1', v2'> ...}
   1.5 Combine (Reduce before shuffle)
2. Shuffle — Put all <k1, v1>, <k2, v2> s.t. k1 = k2 on same machine
3. Reduce
   <k2, v1, v2, v3, ... vm> → {k2, v}

Word Count

Input: large set of documents = { words }

output: For each word: How many times

Map V = {w1, w2, w3 ... wn} → {<w1, 1>, <w2, 1> ...}

Reduce (k=word | v1=1, v2=1, v3=1, ... vk=1}

$\rightarrow \langle word \mid \sum_{i=1}^{n} v_i \rangle$

word = "the" 7% of all words

Combine: $\langle word \mid \sum_{i=1}^{n} v_i \rangle$

"curse of the last reducer"

Inverted Index

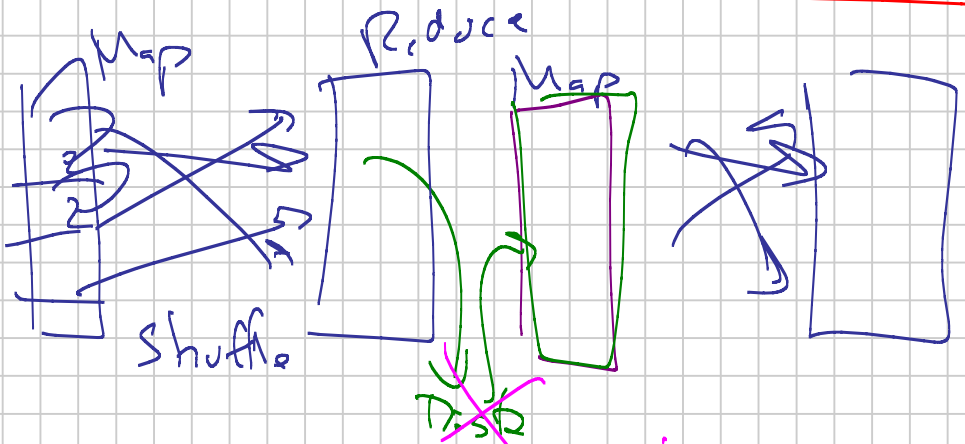
Input: web

Output: <word, list pages>

Map: html $\rightarrow$ { <word$_1$, (p, info)>
<word$_2$, (p, info)>
⋮
<word$_m$, (p, info)> }

Reduce: <word | (p$_1$, info), (p$_2$,), (p$_3$,) ...)
$\rightarrow$ word | sort (p$_1$ | p$_2$ | ... | p$_n$)

Rounds

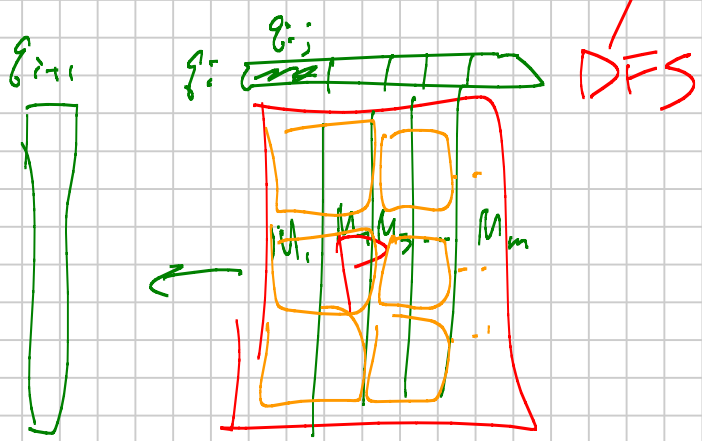


Big overhead

Page Rank on MR

So rounds

$$q_{i+1} = \left((1-\beta)P + \beta \tfrac{1}{n}\right) q_i$$



Map: $r_j = M_j q_{ij}$

Reduce: $q_{i+1} = \sum_j r_j (1-\beta) + \frac{\beta}{n}$