Load

big data technology

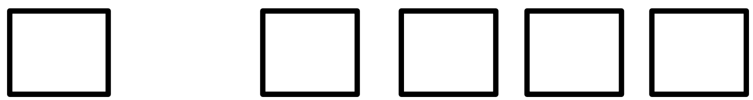
week 3:

map-reduce and programming assignment

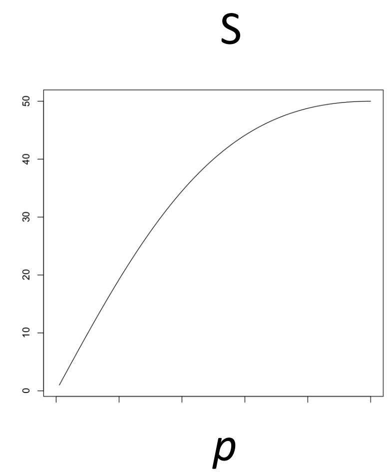
week 4:

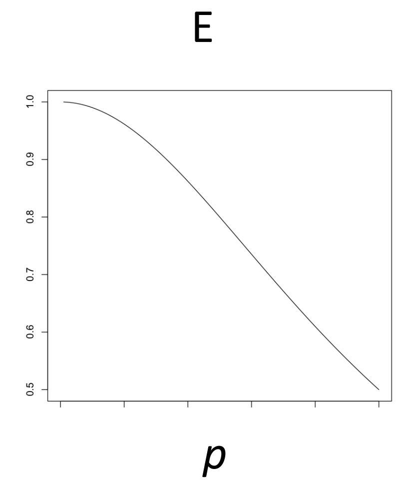
distributed file-systems, databases, and trends

parallel computing



speedup, $S = T_1 / T_p$, time with *p* processors vs with one

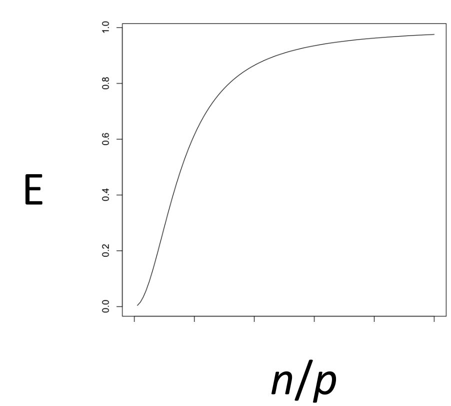




efficiency, $E = T_1 / p\ T_p$

scalable algorithm – E increasing function of *n*/*p*

where *n* is 'problem size'



parallel programming paradigms

shared memory

- partition work

$F(w_p)$:

shared a

lock(a[i])

work(w_p)

unlock(a[i])

message passing

- partition data

$F(p)$:

$a_p = a[p \ldots p+(n/p)-1]$

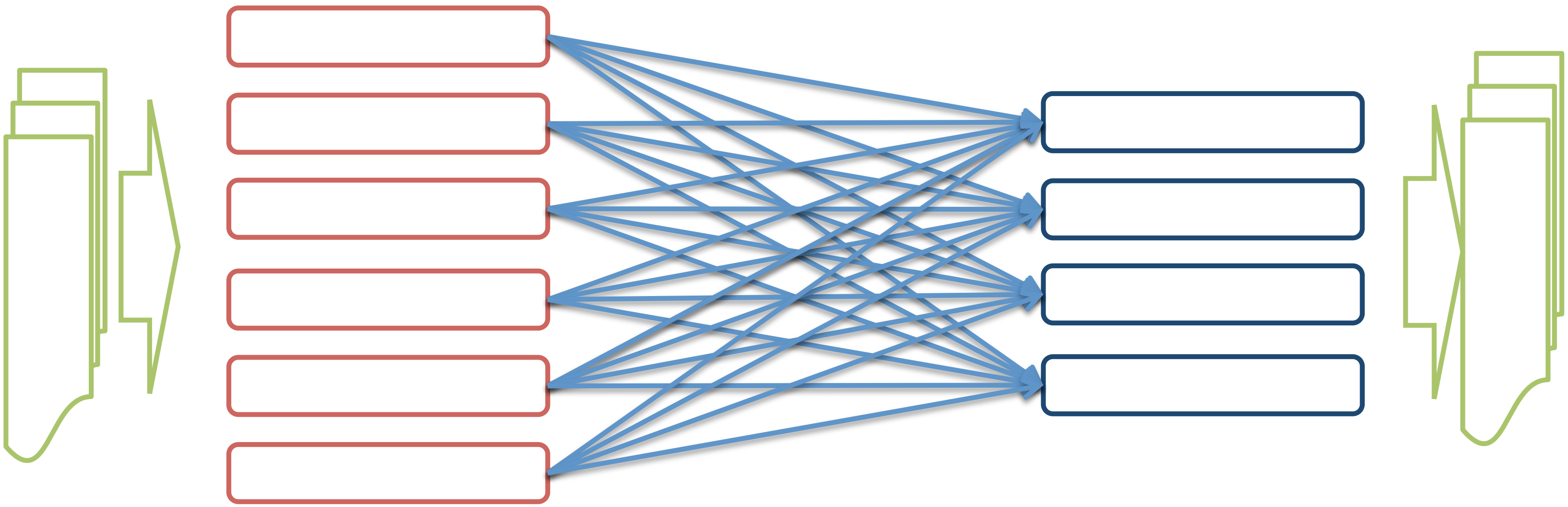
work(w)

exchange data(a_p)

shared + partition data; message-passing + partition work also possible

map-reduce: message-passing, data-parallel, pipelined work, higher level

map-reduce



mappers:
take in k1, v1 pairs
emit k2, v2 pairs
k2,v2 <- map(k1,v1)

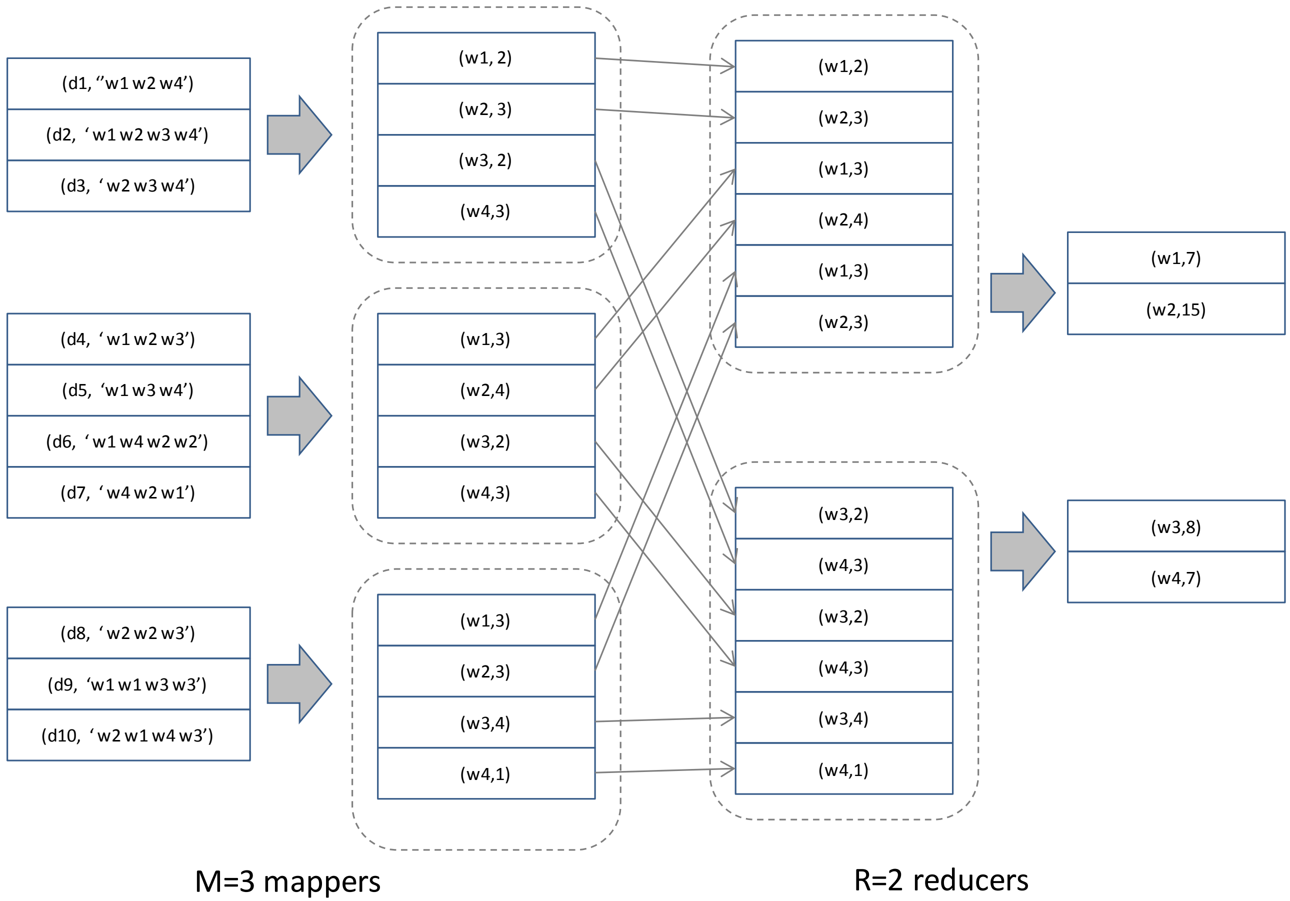
reducers:
receive all pairs for some k2
combine these in some manner
k2,f_r(…v2….) <- reduce(k2, […v2…])

map-reduce platform responsible for *routing* pairs to reducers
map-reduce reads data and *writes* fresh data; is a *batch* process

map-reduce

Map: $(d_k, `w_1 \ldots w_n') \rightarrow [(w_i, c_i)]$ document -> word-count pairs

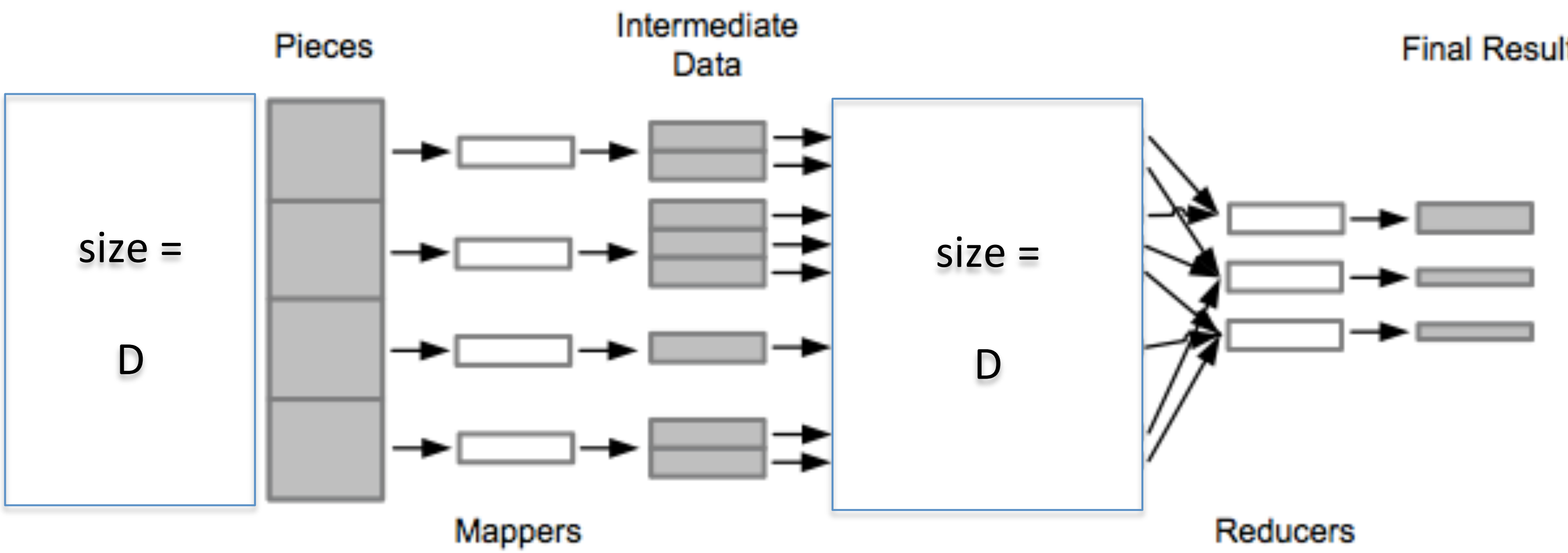
Reduce: $(w_i, [c_i]) \rightarrow (w_i, \sum_i c_i)$ word, count-list -> word-count-total



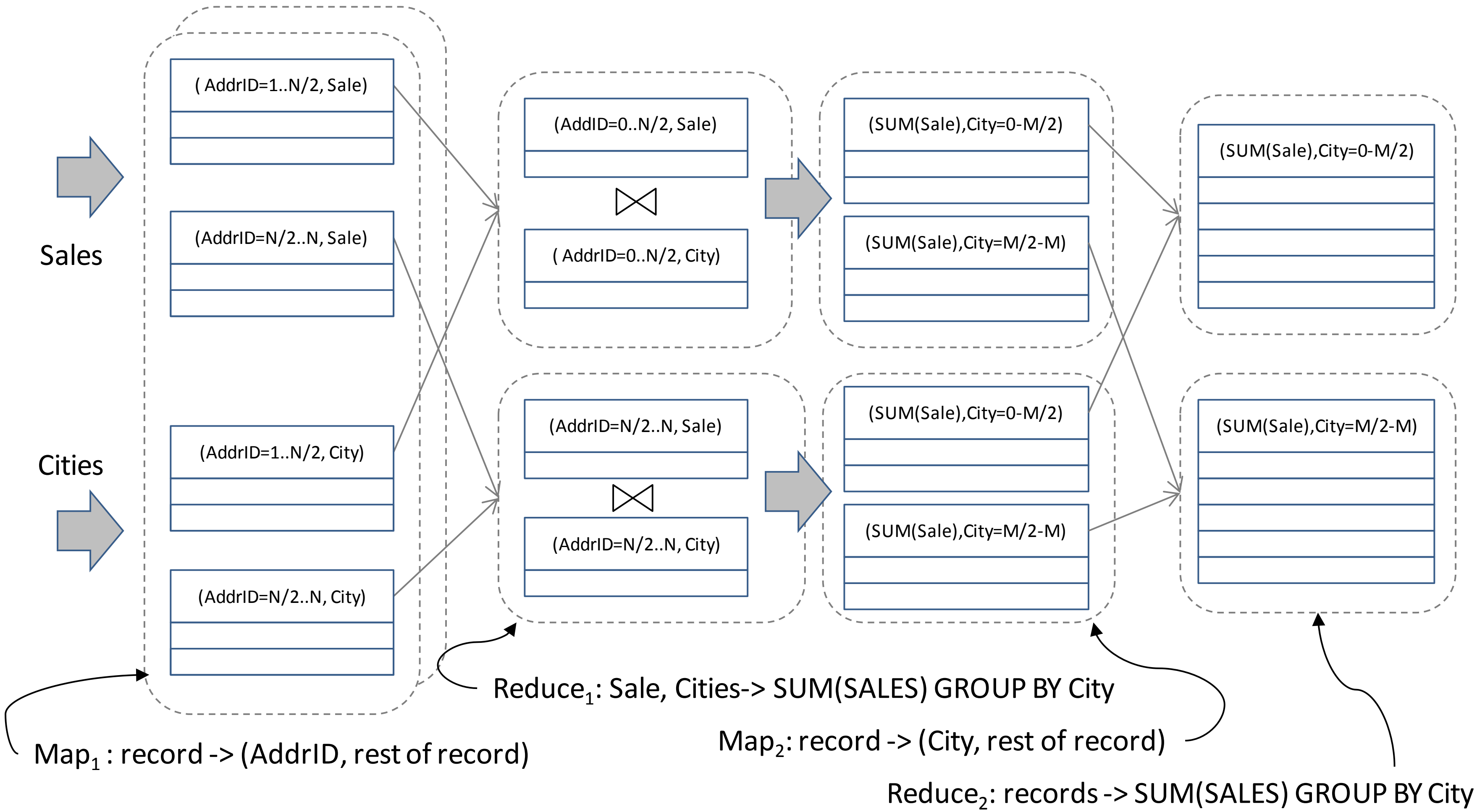
map, reduce … also ‘combine’

how much data is produced by map?

- each word is emitted multiple times!
- combiner : sum up word-counts per mapper before emitting



database join using map-reduce



SQL: SELECT SUM(Sale), City FROM Sales, Cities WHERE Sales.AddrID=Cities.AddrID GROUP BY City

real-world example

lots of data …

paper, author, contents

million such papers, million authors, millions of possible terms (‘phrases’ occurring in contents)

problems:

top 10 terms for each author; top 10 authors per term…

‘database’ person’s solution ….

Q = select id, word, author
from P where in(w,content)

id (paper-id)	content	author

P million

id	word	author

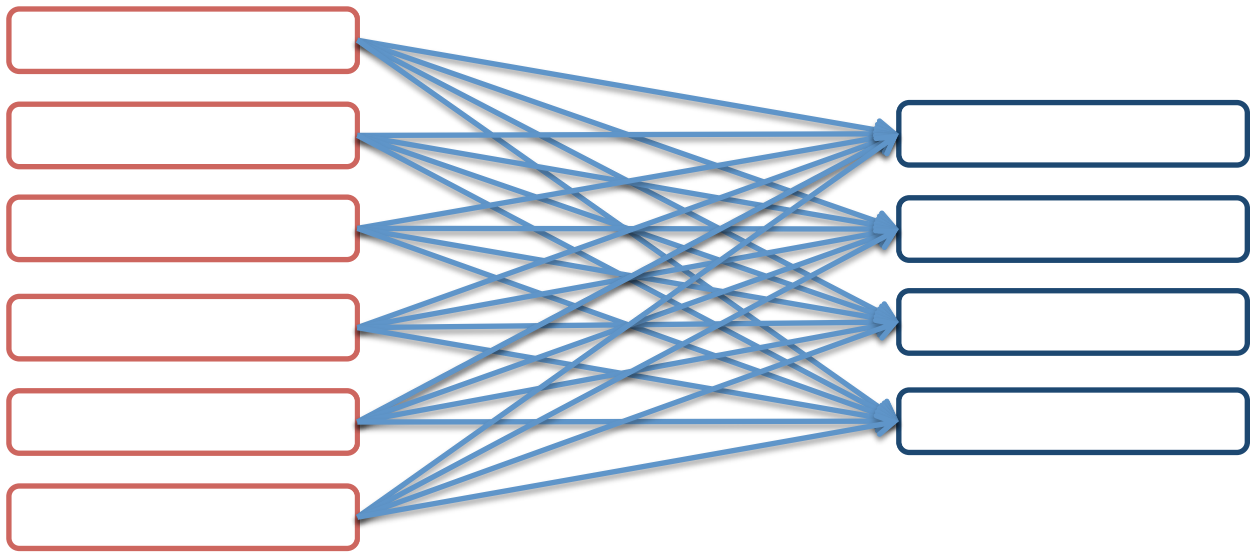
Q

select count(), word, author
from Q group by word

wc	word	author

trillions (million x million)!

top-k words per author in map-reduce



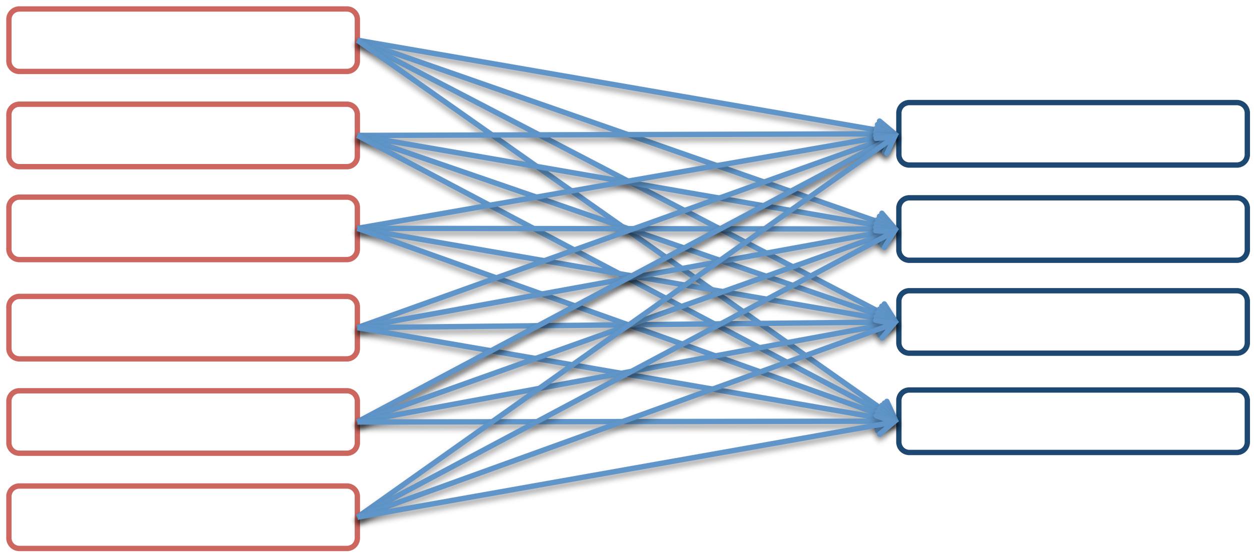
map:
emit word, author

reduce:
reduce-key = word+author
reduce-function = count

suffers from same problem – trillion combinations!

- map-reduce alone is not enough – *approach needs to change!*

top-k words per author in map-reduce



map:
  emit author, contents

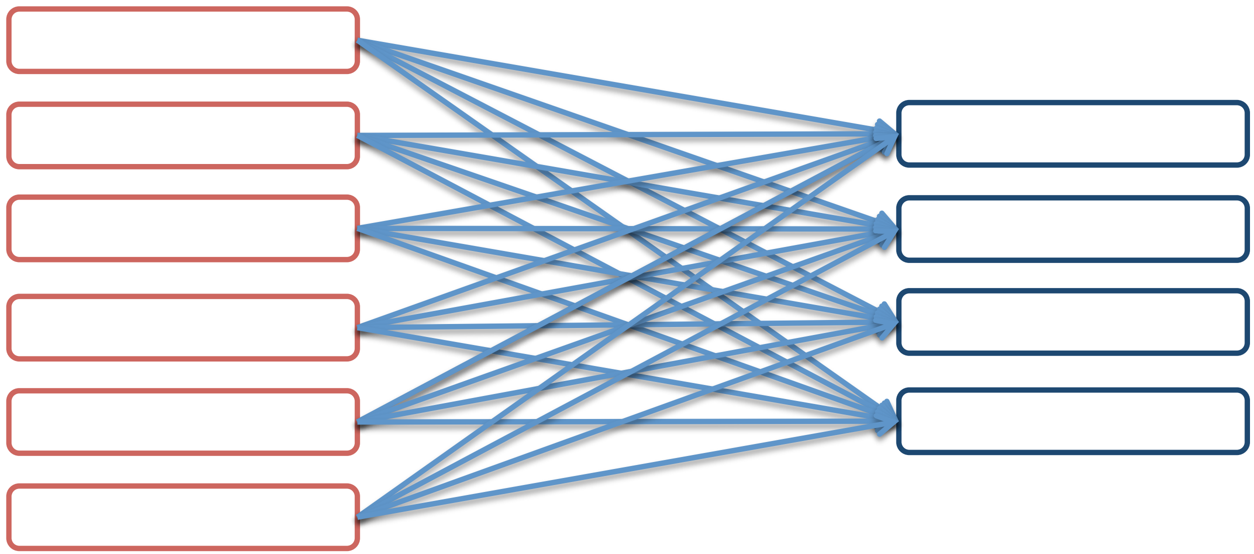
reduce:
  reduce-key = author
  reduce-function = **F()**

F(): for each author:
  scan all inputs and compute word-counts .. insert into w
  sort w, <u>output the top k</u>, delete w and reinitialize to [ ]

look, listen examples in map-reduce

- indexing
- locality-sensitive hashing – how to assemble
- likelihoods – for Bayesian classification
- likelihood ratio – do you need parallelism?
- TF-IDF - HW
- joint probabilities - HW

indexing in map-reduce

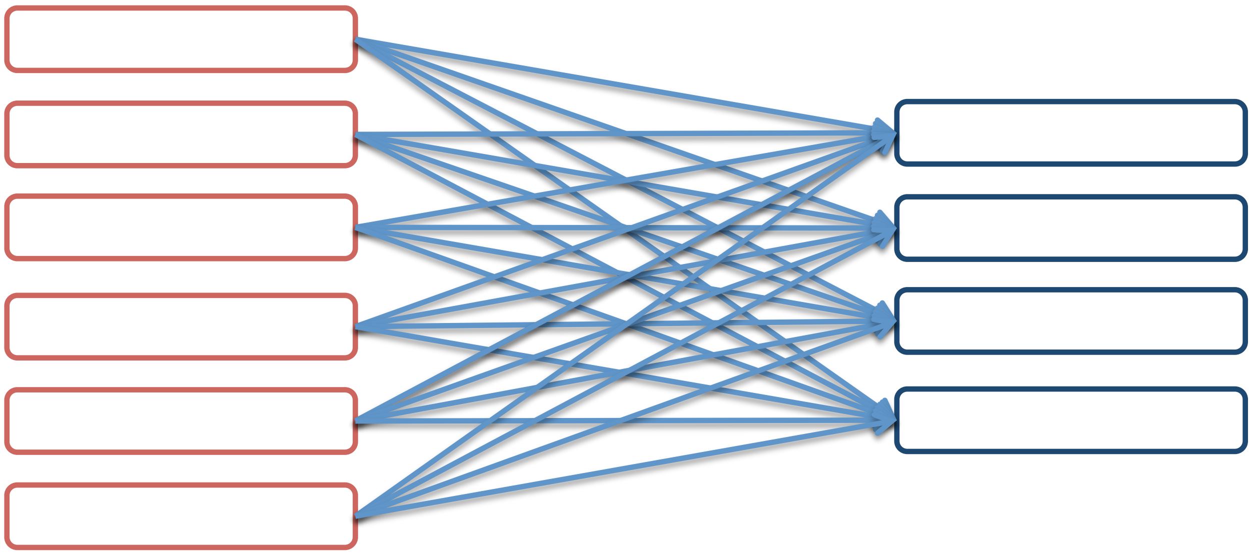


map:
  produce a partial index
  i.e. emit w -> postings-list

reduce:
  reduce-key = word
  merge partial indexes
  i.e. merge postings per word

what about sorting by either document-id, or page-rank etc. ?

LSH in map-reduce

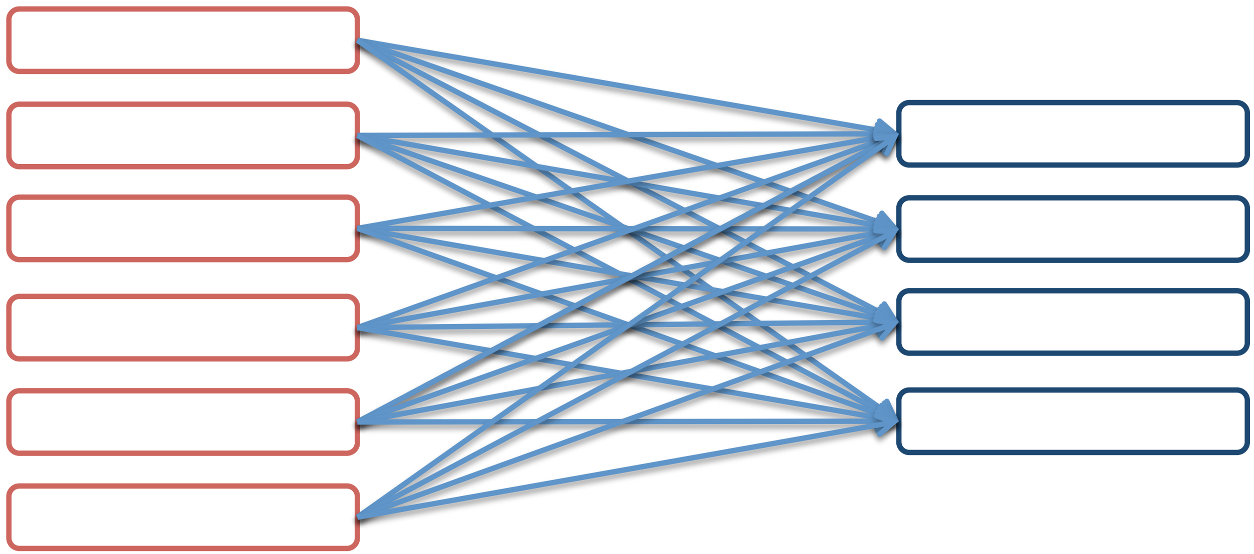


map:
emit doc-id, k hash-values

reduce:
reduce-key = hashes
emit doc-pairs for each key

will a document-pair be emitted by more than one reducer?

likelihoods in map-reduce

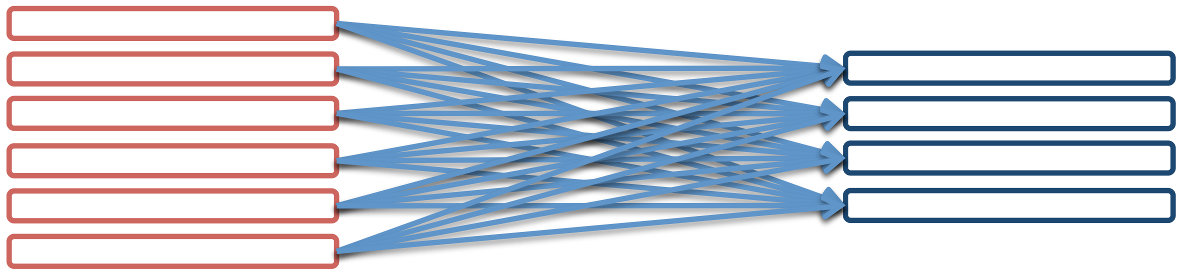


map:
emit counts (f, yes), (f, no)

reduce:
reduce-key = features
sum the counts, divide by N_f
emit the log-likelihoods

once we have the log-likelihoods for each features, do we need parallelism for testing new documents using naïve Bayes?

parallel efficiency of map-reduce



σD data (post map), *P* processors – mappers + reducers
assume *wD* is the useful work needs to be done.

Overheads: $\frac{\sigma D}{P}$ intermediate data is written by each mapper

the time for transmitting it to *P* reducers: $\frac{\sigma D}{P^2} \times P = \frac{\sigma D}{P}$

$$\epsilon_{MR} = \frac{wD}{P(\frac{wD}{P} + 2c\frac{\sigma D}{P})} = \frac{1}{1 + \frac{2c}{w}\sigma}$$

scalable: efficiency approaches 1 as useful work per data-item *w* grows, <u>independent</u> of *P*

parallel-efficiency of MR word-counting

n documents, *m* words, occurring *f* times per document on average, so $D = nmf$

the map phase produces mP partial counts,

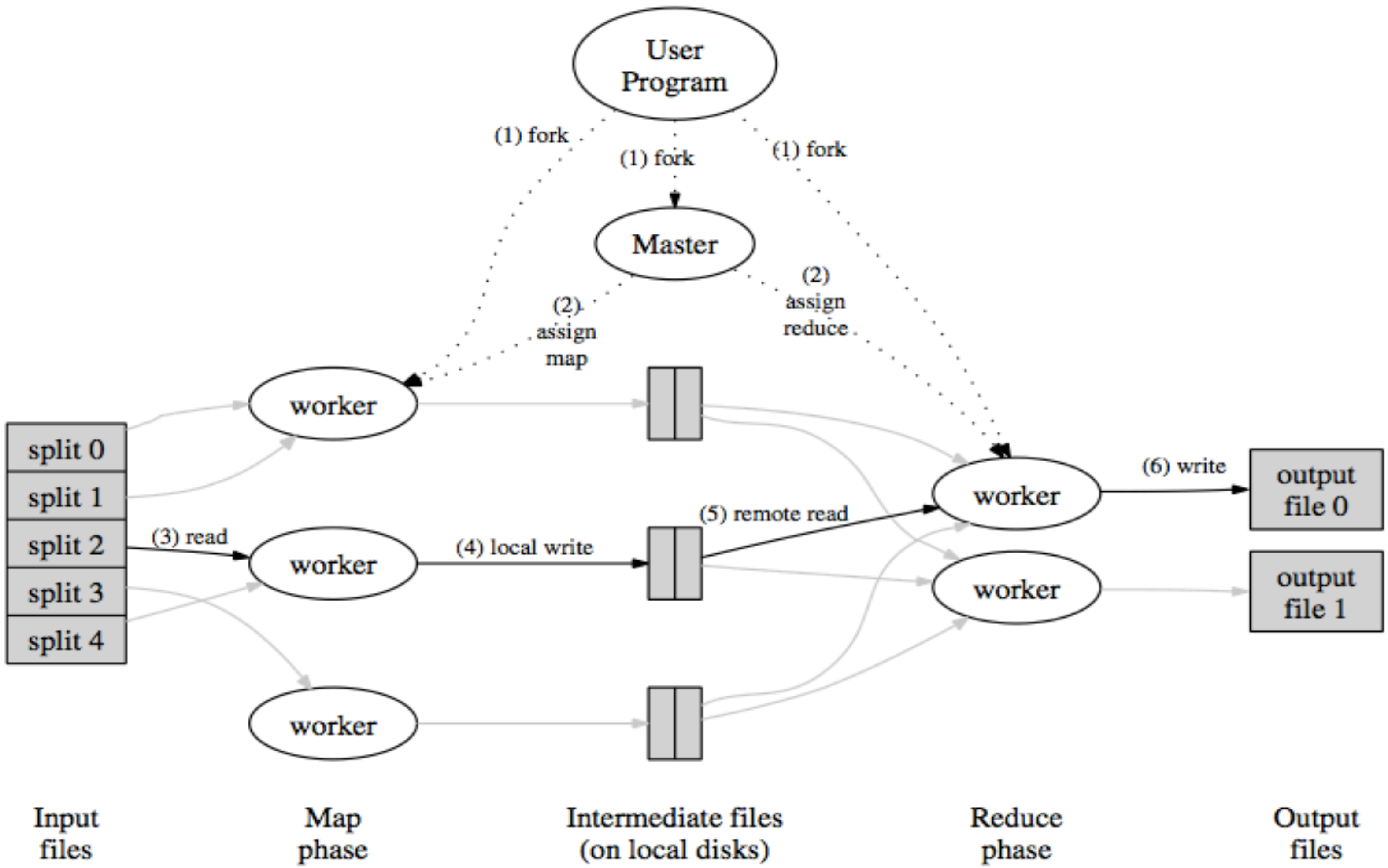
$$\sigma = \frac{mP}{nmf} = \frac{P}{nf}$$

and

$$\varepsilon_{MR} = \frac{1}{1+\frac{2cP}{wnf}} = \frac{1}{1+\frac{2P}{nf}}$$

now, scalability is evident as $\frac{n}{p} \rightarrow \infty$

inside map-reduce



recap and preview

parallel computing

map-reduce, applications, internals

Next week:

distributed file systems

distributed (no-SQL) databases

emerging trends