

# Computational Frameworks

## MapReduce

# Big data challenges

## Computational complexity:

- Any processing requiring a *superlinear number of operations* may easily turn out *unfeasible*.
- If input size is really huge, just *touching all data items* is already quite time consuming.  
E.g., consider an index for  $\simeq 50 \cdot 10^9$  web pages each of average size 20KB  $\rightarrow$  1000TB of data.
- For computation-intensive (e.g., optimization) algorithms, exact solutions may be too costly. Need to resort to *accuracy-efficiency tradeoffs*.

## Effective use of parallel/distributed platforms:

- Specialized high-performance architectures are costly and become rapidly obsolete
- Fault-tolerance becomes serious issue: *low Mean-Time Between Failures* (MTBF).
- Parallel/distributed programming requires *high skills*

# MapReduce

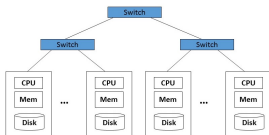
- Introduced by Google in 2004 (see [DG08])
- Programming framework for handling **big data**
- Employed in **many application scenarios** on **clusters of commodity processors** and **cloud infrastructures**
- Main features:
  - Data centric view
  - Inspired by functional programming (map, reduce functions)
  - Ease of programming. Messy details (e.g., **task allocation**; **data distribution**; **fault-tolerance**; **load-balancing**) are hidden to the programmer
- Main implementation: Apache Hadoop



- Hadoop ecosystem: several variants and extensions aimed at improving Hadoop's performance (e.g., Apache Spark)

# Typical cluster architecture

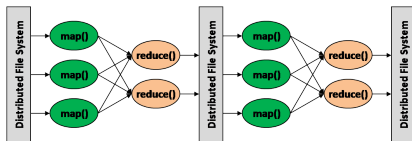
- Racks of 16-64 compute nodes (commodity hardware), connected (within each rack and among racks) by fast switches (e.g., 10 Gbps Ethernet)



- Distributed File System
  - Files divided into *chunks* (e.g., 64MB per chunk)
  - Each chunk replicated (e.g., 2x or 3x) with replicas in different nodes and, possibly, in different racks
  - The distribution of the chunks of a file is represented into a *master node* file which is also replicated. A directory (also replicated) records where all master nodes are.
  - Examples: Google File System (GFS); Hadoop Distributed File System (HDFS)

# MapReduce computation

- Computation viewed as a **sequence of rounds**. (In fact, the original formulation considered only one round)
- Each round transforms a set of **key-value pairs** into another set of **key-value pairs** (data centric view!), through the following two phases
  - **Map phase**: a user-specified **map function** is applied separately to each input key-value pair and produces  $\geq 0$  other key-value pairs, referred to as **intermediate key-value pairs**.
  - **Reduce phase**: the intermediate key-value pairs are **grouped by key** and a user-specified **reduce function** is applied separately to each group of key-value pairs with the same key, producing  $\geq 0$  other key-value pairs, which is the output of the round.
- The output of a round is the input of the next round:



## MapReduce round

- Input file is split into  $X$  chunks and each chunk forms the input of a map task.
- Each map task is assigned to a worker (a compute node) which applies the map function to each key-value pair of the corresponding chunk, buffering the intermediate key-value pairs it produces in its local disk
- The intermediate key-values pairs are partitioned into  $Y$  buckets (key  $k \rightarrow$  bucket  $\text{hash}(k) \bmod Y$ ) and each bucket forms the input of a different reduce task. Note that at the end of the map phase, the key-values pairs constituting a bucket are spread among several disks.
- Each reduce task is assigned to a worker (a compute node), which sorts the key-value pairs of the corresponding bucket by key, and applies the reduce function to each group of key-value pairs with the same key (represented as a key with a list of values), writing the output on the DFS. The application of the reduce function to one group is referred to as a reducer

## MapReduce round (cont'd)

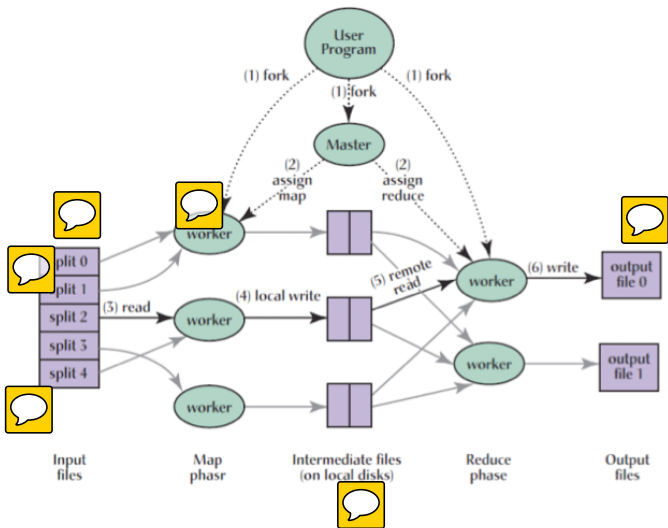
- The user program is forked into a **master** process and several **worker** processes. The master is in charge of assigning map and reduce tasks to the various workers, and to monitor their status (idle, in-progress, completed).
- Input and output files reside on a Distributed File System, while intermediate data are stored on the local disks of the workers
- The round involves a **data shuffle** for moving the intermediate key-value pairs from the compute nodes where they were produced (by map tasks) to the compute nodes where they must be processed (by reduce tasks).

**N.B.: Typically, most expensive operation of the round**

- The values **X** and **Y** can be defined by the user

# MapReduce round (cont'd)

From the original paper:





## Dealing with faults

- The Distributed File System is fault-tolerant
- Master pings workers periodically to detect failures
- Worker failure:
  - Map tasks completed or in-progress at failed worker are reset to idle and will be rescheduled. Note that even if a map task is completed, the failure of the worker makes its output unavailable to reduce tasks, hence it must be rescheduled.
  - Reduce tasks in-progress at failed worker are reset to idle and will be rescheduled.
- Master failure: the whole MapReduce task is aborted

# Specification of a MapReduce algorithm

A *MapReduce (MR) algorithm* should be specified so that

- The input and output of the algorithm is clearly defined
- The **sequence of rounds** executed for any given input instance can be easily constructed
- **For each round** the following aspects are clear
  - input, intermediate and output sets of key-value pairs
  - functions applied in the map and reduce phases.
- Meaningful values (or asymptotic) bounds for the key performance indicators (defined later) can be derived.

## Specification of a MapReduce algorithm (cont'd)

For example, an MR algorithm with a fixed number of rounds  $R$ , we can use the following style:

**Input:** description of the input as set of key-value pairs

**Output:** description of the output as set of key-value pairs

### Round 1:

- *Map phase*: description of the function applied to each key-value pair
- *Reduce phase*: description of the function applied to each group of key-value pairs with the same key

**Round 2:** as before ..

...

**Round R:** as before ...

# Analysis of a MapReduce algorithm

The analysis of an MR algorithm aims at estimating the following key performance indicators (see [P+12]):

- Number of rounds  $R$
- Local space  $M_L$ : maximum amount of space required by *any* map or reduce function executed by the algorithm for storing input and temporary data (but not the output of the function)
- Aggregate space  $M_A$ : maximum amount of space which, at any time during the execution of the algorithm, is needed to store all data required at that time or at future times.

## Observations:

- The indicators are usually estimated through asymptotic analysis as functions of the instance parameters (e.g., size). The analysis could be worst-case or probabilistic.
- In general, the number of rounds  $R$  depends on the instance, on  $M_L$ , and on  $M_A$ . Typically, the larger  $M_L$  and  $M_A$ , the smaller  $R$ .

# Design goals for MapReduce algorithms

The following theorem encapsulates a simple observation.

## Theorem

*For every computational problem solvable sequentially with space complexity  $S(|input|)$  there exists a 1-round MapReduce algorithm with  $M_L = M_A = \Theta(S(|input|))$*

However, the trivial solution implied by the above theorem is impractical for large inputs.

## Design goals for MapReduce algorithms (cont'd)

### Design Goals

**For efficiency, the design of an MR algorithm should aim at:**

- Few rounds (e.g.,  $R = O(1)$ )
- Sublinear local space (e.g.,  $M_L = O(|input|^\epsilon)$ , for some constant  $\epsilon \in (0, 1)$ )
- Linear aggregate space (i.e.,  $M_A = O(|input|)$ ), or only slightly superlinear
- Polynomial complexity of each map or reduce function .

## Observations

- The domain of the keys (resp., values) in input to a map or reduce function may be different from the domain of the keys (resp., values) produced by the function. Also, the reduce phase of a round, can be merged with the map phase of the following round.
- Besides the technicality of representing data as key-value pairs (which is sometimes omitted, when easily derivable) an MR algorithm aims at *breaking the computation into a (hopefully small) number of iterations that execute several tasks in parallel, each task working on a (hopefully small) subset of the data.*
- The round complexity metric is somewhat rough since it ignores the runtimes of the map and reduce functions and the actual volume of data shuffled at each round. More sophisticated metrics exist but are less usable.
- **Curse of the last reducer:** in some cases, one, or a few reducers may be much slower than the other ones, thus delaying the end of the round. When designing MapReduce algorithms one should try to ensure some load balancing (if at all possible) among reducers.

## Basic primitives and techniques



## Word count

**Input:** collection of text documents  $D_1, D_2, \dots, D_k$  containing  $N$  words overall (counting repetitions). Each document is a key-value pair, whose key is the document's name and the value is its content.

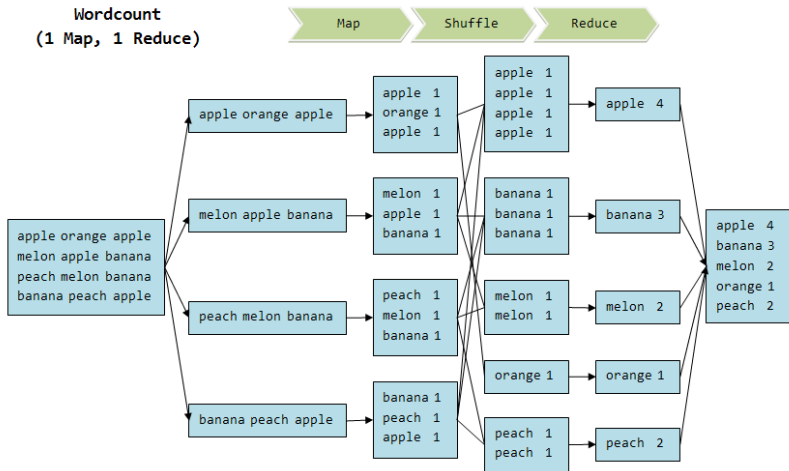
**Output:** The set of pairs  $(w, c(w))$  where  $w$  is a word occurring in the documents, and  $c(w)$  is the number of occurrences of  $w$  in the documents.

### Round 1:

- *Map phase:* for each document  $D_i$ , produce the set of intermediate pairs  $(w, 1)$ , one for each occurrence of a word  $w \in D_i$ . **N.B.:** the word is the key of the pair.
- *Reduce phase:* For each word  $w$ , gather all intermediate pairs  $(w, 1)$  and produce the pair  $(w, c(w))$  by summing all values (1's) of these pairs.

# Example

Wordcount  
(1 Map, 1 Reduce)



## Analysis of Word count

Worst-case analysis with respect to input size  $N$  (= aggregate number of word occurrences)

- $R = 1$
- $M_L = O(N)$ . Bad case: only one word occurs repeated  $N$  times over all documents.
- $M_A = O(N)$

**Observation:** The algorithm does not satisfy the aforementioned design goals: in particular, it does not attain sublinear local space.

## Improved Word count 1

Simple optimization to reduce the space requirements: for each  $D_i$ , the map function produces one pair  $(w, c_i(w))$  for each word  $w \in D_i$ , where  $c_i(w)$  is the number of occurrences of  $w$  in  $D_i$ .

Let  $N_i$  be the number of words in  $D_i$  ( $\Rightarrow N = \sum_{i=1}^k N_i$ ). The optimization yields:

- $R = 1$
- $M_L = O(\max_{i=1,k} N_i + k)$ .
- $M_A = O(N)$

### Observations:

- The sublinear local space requirement is satisfied as long as  $N_i = o(N)$ , for each  $i$ , and  $k = o(N)$ .
- By treating each document as an individual key-value pair we have that for any algorithm  $M_L = \Omega(\max_{i=1,k} N_i)$ .  
 $\Rightarrow$  only the  $O(k)$  additive term can be removed

## Partitioning technique

When some aggregation functions (e.g., sum of the  $c_i(w)$ 's for every word  $i$ ) may potentially receive large inputs (e.g., large  $k$ ) or skewed ones, it is advisable to partition the input, either deterministically or randomly, and perform the aggregation in stages.

We will see two examples:

- An improved version of Word count
- A general Category counting primitive

## Improved Word count 2

Suppose that the Word count problem must be solved for instances consisting of huge numbers of small documents (extreme case  $k = O(N)$ ). The following algorithm features better spaces requirements with respect to the previous one

**Idea:** **partition** intermediate pairs randomly in  $o(N)$  groups and compute counts in two stages

### Round 1:

- *Map phase:* for each document  $D_i$ , produce the intermediate pairs  $(x, (w, c_i(w)))$ , one for every word  $w \in D_i$ , where  $x$  (the key of the pair) is a random value in  $[0, \sqrt{N})$  and  $c_i(w)$  is the number of occurrences of  $w$  in  $D_i$ .
- *Reduce phase:* For each key  $x$  gather all pairs  $(x, (w, c_i(w)))$ , and for each word  $w$  occurring in these pairs produce the pair  $(w, c(x, w))$  where  $c(x, w) = \sum_{(x, (w, c_i(w)))} c_i(w)$ . Now,  $w$  is the key for  $(w, c(x, w))$ .

## Improved Word Count 2 (cont'd)

### Round 2:

- *Map phase*: identity function
- *Reduce phase*: for each word  $w$ , gather the at most  $\sqrt{N}$  pairs  $(w, c(x, w))$  resulting at the end of the previous round, and produce the pair  $(w, \sum_x c(x, w))$ .

**Analysis.** Let  $m_x$  be the number of intermediate pairs with key  $x$  produced by the Map phase of Round 1, and let  $m = \max_x m_x$ . As before, let  $N_i$  be the number of words in  $D_i$ . We have

- $R = 2$
- $M_L = O\left(\max_{i=1,k} N_i + m + \sqrt{N}\right)$ .
- $M_A = O(N)$

**How large can  $m$  be?** We need some technical tool.

## Technical tool: Chernoff bound

### Chernoff bound

Let  $X_1, X_2, \dots, X_n$  be  $n$  i.i.d. Bernoulli random variables, with  $\Pr(X_i = 1) = p$ , for each  $1 \leq i \leq n$ . Thus,  $X = \sum_{i=1}^n X_i$  is a  $\text{Binomial}(n, p)$  random variable. Let  $\mu = E[X] = n \cdot p$ . For every  $\delta_1 \geq 5$  and  $\delta_2 \in (0, 1)$  we have that

$$\Pr(X \geq (1 + \delta_1)\mu) \leq 2^{-(1+\delta_1)\mu}$$

$$\Pr(X \leq (1 - \delta_2)\mu) \leq 2^{-\mu\delta_2^2/2}$$

The proof can be found in [MU05].



## Estimate of $m$ for Word Count 2

### Theorem

*Suppose that the keys assigned to the intermediate pairs in Round 1 are i.i.d. random variables with uniform distribution in  $[0, \sqrt{N})$ . Then, with probability at least  $1 - 1/N^5$*

$$m = O\left(\sqrt{N} \log N\right).$$

Therefore, from the theorem and the preceding analysis we get

$$M_L = O\left(\max_{i=1,k} N_i + \sqrt{N} \log N\right),$$

with probability at least  $1 - 1/N^5$ . In fact, for large  $N$  the probability becomes *very close to 1*.

**N.B.:** This is an example of **probabilistic analysis**, as opposed to more traditional worst-case analysis.


## Estimate of $m$ Word Count 2 (cont'd)

### Proof of theorem.

Let  $N' \leq N$  be the number of intermediate pairs produced by the Map phase of Round 1, and consider an arbitrary key  $x \in [0, \sqrt{N})$ .

**Crucial observation:**  $m_x$  is a  $\text{Binomial}(N', 1/\sqrt{N})$  random variable with expectation  $\mu = N'/\sqrt{N} \leq \sqrt{N}$ .

By the Chernoff bound we have


$$\Pr(m_x \geq 6\sqrt{N} \log N) \leq \frac{1}{2^{6\sqrt{N} \log N}} \leq \frac{1}{N^6}.$$

Now, by union bound we have that

$$\begin{aligned} \Pr(m \geq 6\sqrt{N} \log N) &\leq \sum_{x \in [0, \sqrt{N})} \Pr(m_x \geq 6\sqrt{N} \log N) \\ &\leq \sqrt{N} \Pr(m_x \geq 6\sqrt{N} \log N) \leq \frac{1}{N^5} \end{aligned}$$

Therefore, with probability at least  $1 - 1/N^5$  we have  $m \leq 6\sqrt{N} \log N$ , i.e.,  $m = O(\sqrt{N} \log N)$ . □

## Category counting

Suppose that we are given a set  $S$  of  $N$  objects, each labeled with a category from a given domain, and we want to count how many objects belong to each category.

More precisely:

**Input:** Set  $S$  of  $N$  objects represented by pairs  $(i, (o_i, \gamma_i))$ , for  $0 \leq i < N$ , where  $o_i$  is the  $i$ -th object, and  $\gamma_i$  its category.

**Output:** The set of pairs  $(\gamma, c(\gamma))$  where  $\gamma$  is a category labeling some object of  $S$  and  $c(\gamma)$  is the number of objects of  $S$  labeled with  $\gamma$ .

**Observation:** the straightforward 1-round algorithm may require  $O(N)$  local space, in case a large fraction of objects belong to the same category. In the next slide we will see a more efficient algorithm.

# Category counting (cont'd)

## Round 1:

- *Map phase*: map each pair  $(i, (o_i, \gamma_i))$  into the intermediate pair  $(i \bmod \sqrt{N}, (o_i, \gamma_i))$  (**mod** = remainder of integer division)
- *Reduce phase*: For each key  $j \in [0, \sqrt{N})$  gather the set (say  $S^{(j)}$ ) of all intermediate pairs with key  $j$  and, for each category  $\gamma$  labeling some object in  $S^{(j)}$ , produce the pair  $(\gamma, c_j(\gamma))$ , where  $c_j(\gamma)$  is the number of objects of  $S^{(j)}$  labeled with  $\gamma$ .

## Round 2:

- *Map phase*: identity function
- *Reduce phase*: for each category  $\gamma$ , gather the at most  $\sqrt{N}$  pairs  $(\gamma, c_j(\gamma))$  resulting at the end of the previous round, and produce the pair  $(\gamma, \sum_j c_j(\gamma))$ .

## Exercise

Show that  $R = 2$ ,  $M_L = O(\sqrt{N})$ ,  $M_A = O(N)$ .


## Trading accuracy for efficiency

There are problems for which **exact MR algorithms** may be **too costly**, namely they may require a large number of rounds, or large (i.e., close to linear) local space, or large (i.e., superlinear) aggregate space. These algorithms become impractical for very large inputs.

In these scenarios, **giving up exact solutions (if acceptable for the application) may greatly improve efficiency.**

We'll now see an example through an important primitive for the processing of pointsets from metric spaces (e.g., spatial datasets).

# Maximum pairwise distance

Suppose that we are given a set  $S$  of  $N$  points from some metric space (e.g.,  $\mathbb{R}^3$ ) and we want to determine the maximum distance between two points, for a given distance function  $d(\cdot, \cdot)$ . 

More precisely:

**Input:** Set  $S$  of  $N$  points represented by pairs  $(i, x_i)$ , for  $0 \leq i < N$ , where  $x_i$  is the  $i$ -th point.

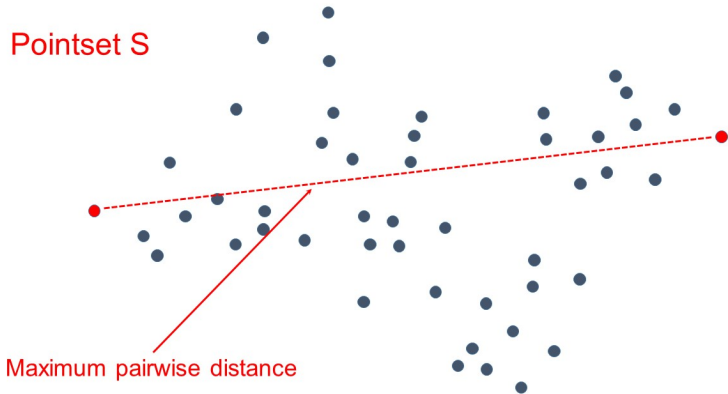
**Output:**  $d_{\max} = \max_{0 \leq i, j < N} d(x_i, x_j)$ .

## Exercise

Design an MR algorithm for the problem which requires  $R = O(1)$ ,  $M_L = O(\sqrt{N})$  and  $M_A = O(N^2)$ .

## Maximum pairwise distance (cont'd)

Pointset S



## Maximum pairwise distance (cont'd)

We can substantially reduce the aggregate space requirements if we tolerate a factor 2 error in the estimate of  $d_{\max}$ . For an arbitrary point  $x_i$  define

$$d_{\max}(i) = \max_{0 \leq j < N} d(x_i, x_j).$$

### Lemma

For any  $0 \leq i < N$  we have  $d_{\max} \in [d_{\max}(i), 2d_{\max}(i)]$ .

### Proof.

It is immediate to see that  $d_{\max} \geq d_{\max}(i)$ . Suppose that  $d_{\max} = d(x_r, x_t)$  for some  $r, t$ . By the triangle inequality we have

$$d(x_r, x_t) \leq d(x_r, x_i) + d(x_i, x_t) \leq 2d_{\max}(i).$$





# Maximum pairwise distance (cont'd)

## Round 1:

- *Map phase*: map each pair  $(i, x_i)$  into the intermediate pair  $(i \bmod \sqrt{N}, (i, x_i))$ . Also, create the  $\sqrt{N} - 1$  pairs  $(j, (0, x_0))$ , for  $1 \leq j < \sqrt{N}$ .
- *Reduce phase*: For each key  $j \in [0, \sqrt{N})$  gather the set  $S^{(j)}$  of all intermediate pairs with key  $j$  (which include  $(j, (0, x_0))$ ) and produce the pair  $(0, d_{\max}(0, j))$  where  $d_{\max}(0, j)$  is the maximum distance between  $x_0$  and the points associated with pairs in  $S^{(j)}$

## Round 2:

- *Map phase*: identity.
- *Reduce phase*: gather all pairs resulting at the end of the previous round, and output  $d_{\max}(0) = \sum_{0 \leq j < \sqrt{N}} d_{\max}(0, j)$ .

**Analysis:**  $R = 2$ ,  $M_L = O(\sqrt{N})$ ,  $M_A = O(N)$ .

## Trading rounds for space efficiency

Minimizing the number of rounds may sometimes force large (i.e., impractical) space requirements. In these cases, one could aim at devising algorithms which feature suitable tradeoffs between local and/or aggregate space requirements and number of rounds.

We already saw an example (Improved Word count 2) of this kind of tradeoffs.

In what follows, we will see further examples considering some basic linear algebra primitives.

# Matrix-vector multiplication

Input:  $n \times n$  matrix  $A$  and  $n$ -vector  $V$

Output:  $W = A \cdot V$

N.B.: heavily used primitive for page-rank computation

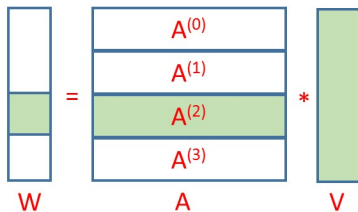
Trivial MapReduce algorithm:

- Let  $A^{(i)}$  denote row  $i$  of  $A$ , for  $0 \leq i < n$ .
- *Map phase*: Create  $n$  replicas of  $V$ : namely  $V_0, V_1, \dots, V_{n-1}$ .
- *Reduce phase*: For every  $0 \leq i < n$  in parallel, compute  $W[i] = A^{(i)} \cdot V_i$

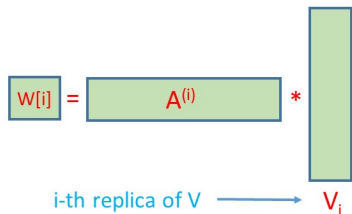
**Analysis:**  $R = 1$  round;  $M_L = O(n)$ ;  $M_A = O(n^2)$  (i.e., sublinear local space and linear aggregate space).

**Exercise:** Specify input, intermediate and output key-value pairs

# Primitives: matrix-vector multiplication (cont'd)



**Round 1**  
for each  $i$  in a  
distinct reducer



## Matrix-vector multiplication (cont'd)

What happens if  $n$  is very large and the available local space is  $M_L = o(n)$ ? Can we trade rounds for smaller local space?

More space-efficient MapReduce algorithm:

- Let  $k = n^{1/3}$  and assume, for simplicity, that  $k$  is an integer and divides  $n$ . Consider  $A$  subdivided into  $(n/k)^2$   $k \times k$  blocks ( $A^{(s,t)}$ , with  $0 \leq s, t < n/k$ ), and  $V$  and  $W$  subdivided into  $n/k$  segments of size  $k$  ( $V^{(t)}$ ,  $W^{(s)}$ , with  $0 \leq t, s < n/k$ ), in such a way that

$$W^{(s)} = \sum_{t=0}^{n/k-1} A^{(s,t)} \cdot V^{(t)}.$$

# Matrix-vector multiplication (cont'd)

- Round 1:

- Map phase:* For every  $0 \leq t < n/k$ , create  $n/k$  replicas of  $V^{(t)}$ . Call these replicas  $V_s^{(t)}$ , with  $0 \leq s < n/k$
- Reduce phase:* For every  $0 \leq s, t < n/k$  in parallel, compute  $W_t^{(s)} = A^{(s,t)} \cdot V_s^{(t)}$

- Round 2:

- Map phase:* identity
- Reduce phase:* For every  $0 \leq i < n$  gather the  $n/k$  contributions to  $W[i]$  and add them up to compute the final value. (If  $W[i]$  belongs to segment  $W^{(s)}$ , the  $n/k$  contributions come from  $W_t^{(s)}$ , for  $0 \leq t < n/k$ .)

**Analysis:**  $R = 2$  rounds;  $M_L = O(k^2 + n/k) = O(n^{2/3})$ ;  $M_A = O(n^2)$ .

**Exercise:** Specify input, intermediate and output key-value pairs

**Observation:** Compared to the trivial algorithm, the local space decreases from  $O(n)$  to  $O(n^{2/3})$ , at the expense of an extra round.

# Matrix-vector multiplication (cont'd)

$$\begin{array}{c}
 \begin{array}{|c|} \hline W^{(0)} \\ \hline \\ \hline W^{(1)} \\ \hline \\ \hline W^{(2)} \\ \hline \\ \hline W^{(3)} \\ \hline \end{array} \\
 W
 \end{array}
 =
 \begin{array}{c}
 \begin{array}{|c|c|c|c|} \hline A^{(0,0)} & A^{(0,1)} & A^{(0,2)} & A^{(0,3)} \\ \hline \\ \hline A^{(1,0)} & A^{(1,1)} & A^{(1,2)} & A^{(1,3)} \\ \hline \\ \hline A^{(2,0)} & A^{(2,1)} & A^{(2,2)} & A^{(2,3)} \\ \hline \\ \hline A^{(3,0)} & A^{(3,1)} & A^{(3,2)} & A^{(3,3)} \\ \hline \end{array} \\
 A
 \end{array}
 *
 \begin{array}{c}
 \begin{array}{|c|} \hline V^{(0)} \\ \hline \\ \hline V^{(1)} \\ \hline \\ \hline V^{(2)} \\ \hline \\ \hline V^{(3)} \\ \hline \end{array} \\
 V
 \end{array}$$

**Round 1**

for each s,t in a distinct reducer

$$W_t^{(s)} = A^{(s,t)} * V_s^{(t)}$$

**Round 2**

For each s in a distinct reducer

$$W^{(s)} = W_0^{(s)} + W_1^{(s)} + W_2^{(s)} + W_3^{(s)}$$

*N.B. The subscript index denotes the replica number*

# Primitives: matrix multiplication

Input:  $n \times n$  matrices  $A, B$

Output:  $C = A \cdot B$

Trivial MapReduce algorithm:

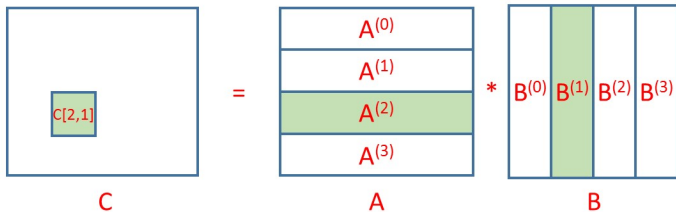
- Let  $A^{(i)}, B^{(j)}$  denote row  $i$  of  $A$  and column  $j$  of  $B$ , respectively, for  $0 \leq i, j < n$ .
- *Map phase*: Create  $n$  replicas of each row  $A^{(i)}$  and of each column  $B^{(j)}$ . Call these replicas  $A_t^{(i)}$  and  $B_t^{(j)}$ , with  $0 \leq t < n$ .
- *Reduce phase*: For every  $0 \leq i, j < n$  compute  $C[i, j] = A_j^{(i)} \cdot B_i^{(j)}$ .

**Analysis:**  $R = 1$  round;  $M_L = O(n)$ ;  $M_A = O(n^3)$  (i.e., sublinear local space but superlinear aggregate space).

**Exercise:** Specify input, intermediate and output key-value pairs



# Matrix multiplication (cont'd)



**Round 1**  
for each  $i, j$  in a distinct reducer

$$C[i,j] = A_j^{(i)} * B_i^{(j)}$$

*N.B. The subscript index denotes the replica number*

## Matrix multiplication (cont'd)

Can we trade rounds for smaller local/aggregate space?

More space-efficient MapReduce algorithm:

- Let  $k = n^{1/3}$  and assume, for simplicity, that  $k$  is an integer and divides  $n$ . Consider  $A, B$  and  $C$  subdivided into  $(n/k)^2$   $k \times k$  blocks  $(A^{(s,t)}, B^{(s,t)}, C^{(s,t)})$ , with  $0 \leq s, t < n/k$ , in such a way that

$$C^{(s,t)} = \sum_{\ell=0}^{n/k-1} A^{(s,\ell)} \cdot B^{(\ell,t)}.$$

## Matrix multiplication (cont'd)

- Round 1:
  - *Map phase*: For every  $0 \leq s, t < n/k$ , create  $n/k$  replicas of  $A^{(s,t)}$  and  $B^{(s,t)}$ . Call these replicas  $A_i^{(s,t)}$  and  $B_i^{(s,t)}$ , with  $0 \leq i < n/k$
  - *Reduce phase*: For every  $0 \leq s, t, \ell < n/k$  in parallel, compute  $C_\ell^{(s,t)} = A_t^{(s,\ell)} \cdot B_s^{(\ell,t)}$
- Round 2:
  - *Map phase*: identity
  - *Reduce phase*: For every  $0 \leq i, j < n$  gather the  $n/k$  contributions to  $C[i, j]$  and add them up to compute the final value. (If  $C[i, j]$  belongs to block  $C^{(s,t)}$ , the  $n/k$  contributions come from  $C_\ell^{(s,t)}$ , for  $0 \leq \ell < n/k$ .)

**Analysis:**  $R = 2$  rounds;  $M_L = O(k^2 + n/k) = O(n^{2/3})$ ;  $M_A = O(n^{8/3})$ .

**Exercise:** Specify input, intermediate and output key-value pairs

**Observation:** Compared to the trivial algorithm, both local and aggregate space decrease by a factor  $O(n^{1/3})$ , at the expense of an extra round.

# Matrix multiplication (cont'd)

$C^{(0,0)}$	$C^{(0,1)}$	$C^{(0,2)}$	$C^{(0,3)}$
$C^{(1,0)}$	$C^{(1,1)}$	$C^{(1,2)}$	$C^{(1,3)}$
$C^{(2,0)}$	$C^{(2,1)}$	$C^{(2,2)}$	$C^{(2,3)}$
$C^{(3,0)}$	$C^{(3,1)}$	$C^{(3,2)}$	$C^{(3,3)}$

**C**

=

$A^{(0,0)}$	$A^{(0,1)}$	$A^{(0,2)}$	$A^{(0,3)}$
$A^{(1,0)}$	$A^{(1,1)}$	$A^{(1,2)}$	$A^{(1,3)}$
$A^{(2,0)}$	$A^{(2,1)}$	$A^{(2,2)}$	$A^{(2,3)}$
$A^{(3,0)}$	$A^{(3,1)}$	$A^{(3,2)}$	$A^{(3,3)}$

**A**

\*

$B^{(0,0)}$	$B^{(0,1)}$	$B^{(0,2)}$	$B^{(0,3)}$
$B^{(1,0)}$	$B^{(1,1)}$	$B^{(1,2)}$	$B^{(1,3)}$
$B^{(2,0)}$	$B^{(2,1)}$	$B^{(2,2)}$	$B^{(2,3)}$
$B^{(3,0)}$	$B^{(3,1)}$	$B^{(3,2)}$	$B^{(3,3)}$

**B**

## Round 1

for each  $s, t, l$  in a distinct reducer

$$C_l^{(s,t)} = A_t^{(s,l)} * B_s^{(l,t)}$$

## Round 2

For each  $s, t$  in a distinct reducer

$$C^{(s,t)} = C_0^{(s,t)} + C_1^{(s,t)} + C_2^{(s,t)} + C_3^{(s,t)}$$

*N.B. The subscript index denotes the replica number*

# Observations

What happens if bounds on  $M_L$  or  $M_A$  are imposed to the algorithm?

Adapt the segment and block sizes to the constraints. For example, for fixed  $M_L$  we suitably set the segment/block sizes and perform the final aggregation in several rounds, if needed. For Matrix-vector multiplication, the following performance can be attained (see [P+12] for details):

- Matrix-vector multiplication:

$$R : O\left(\frac{\log n}{\log M_L}\right)$$

$$M_L : \text{any fixed value}$$

$$M_A : O(n^2) \text{ (MINIMUM!).}$$

$\Rightarrow$  Matrix-vector multiplication can be performed in  $O(1)$  rounds using linear aggregate space as long as  $M_L = \Omega(n^\epsilon)$ , for some constant  $\epsilon \in (0, 1)$ .

## Observations (cont'd)

For Matrix multiplication, smaller block size results in a higher aggregate space. However, the block replications can be orchestrated to control the aggregate space blow-up at the expense of an increased number of rounds. The following performance can be attained (see [P+12]):

- Matrix multiplication:

$$R : O\left(\frac{n^3}{M_A \sqrt{M_L}} + \frac{\log n}{\log M_L}\right)$$

$M_L$  : any fixed value

$M_A$  : any fixed value  $\Omega(n^2)$

$\Rightarrow$  Matrix multiplication can be performed in  $O(1)$  rounds as long as  $M_L = \Omega(n^\epsilon)$ , for some constant  $\epsilon \in (0, 1)$ , and  $M_A \sqrt{M_L} = \Omega(n^3)$ .

The total number of operations executed by the above algorithms (referred to as **work**) is asymptotically the same as the one of the straightforward sequential algorithms.

## Exploiting samples

The first question that should be asked when facing a big-data processing task is the following

Can small subsets of the data help speeding up the task?

In many cases, the answer is **YES!** Specifically, a small sample, suitably extracted, could be exploited for the following purposes.

- To subdivide the dataset in smaller subsets to be analyzed separately.
- To provide a succinct yet accurate representation of the whole dataset, which contains a good solution to the problem and filters out noise and outliers, thus allowing the execution of the task on the sample.

In what follows, we will see an example of the first type thorough the **sorting primitive**.

# Sorting

**Input:** Set  $S = \{s_i : 0 \leq i < N\}$  of  $N$  distinct sortable objects (each  $s_i$  represented as a pair  $(i, s_i)$ )

**Output:** Sorted set  $\{(i, s_{\pi(i)}) : 0 \leq i < N\}$ , where  $\pi$  is a permutation such that  $s_{\pi(1)} \leq s_{\pi(2)} \leq \dots \leq s_{\pi(N)}$ .

**MR SampleSort** algorithm. The algorithm is based on the following idea:



- Fix a suitable integral design parameter  $K$ .
- Randomly select some objects ( $K$  on average) as splitters.
- Partition the objects into subsets based on the ordered sequence of splitters.
- Sort each subset separately and compute the final ranks.



# MR SampleSort

## Round 1:

- *Map phase*: for each pair  $(i, s_i)$  do the following: create the intermediate pair  $(i \bmod K, (0, s_i))$  and, with probability  $p = K/N$  independently of other objects, select  $s_i$  as a **splitter**. If  $s_i$  is selected as splitter then create  $K$  additional intermediate pairs (*splitter pairs*)  $(j, (1, s_i))$ , with  $0 \leq j < K$ . (Note the binary flag used to distinguish splitter pairs from the other intermediate pairs.)

Suppose that  $t$  objects are selected as splitters.

- *Reduce phase*: for  $0 \leq j < K$  do the following: gather all intermediate and splitter pairs with key  $j$ ; sort the splitters (let  $x_1 \leq x_2 \leq \dots \leq x_t$  be the splitters in sorted order and define  $x_0 = -\infty$  and  $x_{t+1} = \infty$ ); for every  $0 \leq \ell \leq t$  and every intermediate pair  $(j, (0, s))$ , with  $x_\ell < s \leq x_{\ell+1}$ , produce the pair  $(\ell, s)$  with key  $\ell$ .

## MR SampleSort (cont'd)

- Round 2:
  - *Map phase*: identity
  - *Reduce phase*: for every  $0 \leq \ell \leq t$  gather, from the output of the previous round, the set of pairs  $S^{(\ell)} = \{(\ell, s)\}$ , compute  $N_\ell = |S^{(\ell)}|$ , and create  $t + 1$  replicas of  $N_\ell$  (use suitable pairs for these replicas).
- Round 3:
  - *Map phase*: identity
  - *Reduce phase*: for every  $0 \leq \ell \leq t$  do the following: gather  $S^{(\ell)}$  and the values  $N_0, N_1, \dots, N_t$ ; sort  $S^{(\ell)}$ ; and compute the final output pairs for the objects in  $S^{(\ell)}$  whose ranks start from  $1 + \sum_{h=0}^{\ell-1} N_h$ .

## Example

$$N = 32, K = 4$$

$$S = 16, 32, 1, 15, 14, 7, 28, 20, 12, 3, 29, 17, 11, 10, 8, 2, \\ 25, 21, 13, 5, 19, 23, 30, 26, 31, 22, 9, 6, 27, 24, 4, 18$$

**Round 1.** Call  $S_j$  the set of intermediate pairs  $(j, (0, s))$  after the Map phase. We have (objects only):

$$S_0 = 16, 32, 1, 15, 14, 7, 28, 20$$

$$S_1 = 12, 3, 29, 17, 11, 10, 8, 2$$

$$S_2 = 25, 21, 13, 5, 19, 23, 30, 26$$

$$S_3 = 31, 22, 9, 6, 27, 24, 4, 18$$

The  $t = 5$  splitters are highlighted in blue. In sorted order:

$$x_1 = 4, x_2 = 9, x_3 = 16, x_4 = 21, x_5 = 29.$$

## Example (cont'd)

Round 1 (cont'd). Call  $S_j^{(\ell)}$  the set of intermediate pairs  $(j, (0, s))$  with  $x_\ell < s \leq x_{\ell+1}$ . We have (objects only):

$j$	$S_j^{(0)}$	$S_j^{(1)}$	$S_j^{(2)}$	$S_j^{(3)}$	$S_j^{(4)}$	$S_j^{(5)}$
0	1	7	16,15,14	20	28	32
1	3,2	8	12,11,10	17	29	
2		5	13	21,19	25,23,26	30
3	4	9,6		18	22,27,24	31

Round 2 The  $S^{(\ell)}$ 's (objects only) and the  $N_\ell$ 's values are

$$\begin{array}{ll}
 S^{(0)} &= 1, 3, 2, 4 & N_0 &= 4 \\
 S^{(1)} &= 7, 8, 5, 9, 6 & N_1 &= 5 \\
 S^{(2)} &= 16, 15, 14, 12, 11, 10, 13 & N_2 &= 7 \\
 S^{(3)} &= 20, 17, 21, 19, 18 & N_3 &= 5 \\
 S^{(4)} &= 28, 29, 25, 23, 26, 22, 27, 24 & N_4 &= 8 \\
 S^{(5)} &= 32, 30, 31 & N_5 &= 3
 \end{array}$$

## Example (cont'd)

### Round 3. Final output

$(1, 1), \dots, (4, 4)$	from rank 1
$(5, 5), \dots, (9, 9)$	from rank $N_0 + 1 = 5$
$(10, 10), \dots, (16, 16)$	from rank $N_0 + N_1 + 1 = 10$
$(17, 17), \dots, (21, 21)$	from rank $N_0 + N_1 + N_2 + 1 = 17$
$(22, 22), \dots, (29, 29)$	from rank $N_0 + N_1 + N_2 + N_3 + 1 = 22$
$(30, 30), \dots, (32, 32)$	from rank $N_0 + \dots + N_4 + 1 = 30$

# Analysis of MR SampleSort

- Number of rounds:  $R = 3$
- Local Space  $M_L$ :
  - Round 1:  $O(t + N/K)$ , since each reducer must store all splitter pairs and a subset of  $N/K$  intermediate pairs.
  - Round 2:  $O(\max\{N_\ell; 0 \leq \ell \leq t\})$  since each reducer must gather one  $S^{(\ell)}$ .
  - Round 3:  $O(t + \max\{N_\ell; 0 \leq \ell \leq t\})$ , since each reducer must store all  $N_\ell$ 's and one  $S^{(\ell)}$ .

$\Rightarrow$  overall  $M_L = O(t + N/K + \max\{N_\ell; 0 \leq \ell \leq t\})$
- Aggregate Space  $M_A$ :  $O(N + t \cdot K + t^2)$ , since in Round 1 each splitter is replicated  $K$  times, and in Round 3 each  $N_\ell$  is replicated  $t + 1$  times. The objects are never replicated.

# Analysis of MR SampleSort (cont'd)

## Lemma

With reference to the MR SampleSort algorithm, for any  $K \in (2 \ln N, N)$  the following two inequalities hold with high probability (i.e., probability at least  $1 - 1/N$ ):

- ①  $t \leq 6K$ , and
- ②  $\max\{N_i ; 0 \leq \ell \leq t\} \leq 4(N/K) \ln N$ .

## Proof.

Deferred. □

## Theorem

By setting  $K = \sqrt{N}$ , MR SampleSort runs in 3 rounds, and, with high probability, it requires local space  $M_L = O(\sqrt{N} \ln N)$  and aggregate space  $M_A = O(N)$ .

## Proof.

Immediate from lemma. □

# Analysis of MR SampleSort (cont'd)

## Proof of Lemma

We show that each inequality holds with probability at least  $1 - 1/(2N)$ .

- 1  $t$  is a Binomial( $N, K/N$ ) random variable with  $E[t] = K > 2 \ln N$ . By the Chernoff bound (first inequality with  $\delta_1 = 5$ ) we have that  $t > 6K$  with probability at most  $1/(2N)$ .
- 2 View the *sorted* sequence of objects as divided into  $K/(2 \ln N)$  contiguous segments of length  $N' = 2(N/K) \ln N$  each, and consider one arbitrary such segment. The probability that no splitter is chosen among the objects of the segment is

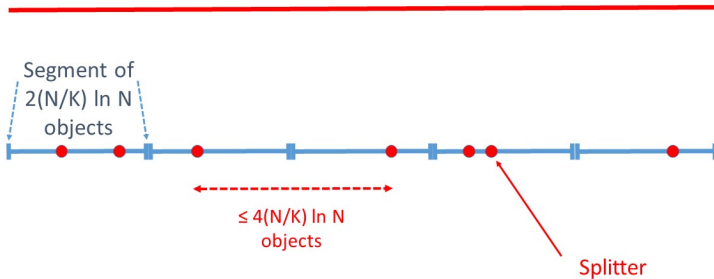
$$\left(1 - \frac{K}{N}\right)^{N'} = \left(1 - \frac{1}{(N/K)}\right)^{(N/K)2 \ln N} \leq \left(\frac{1}{e^{\ln N}}\right)^2 = \frac{1}{N^2}.$$

(We used the well-known inequality  $(1 - 1/x)^x \leq 1/e$ ,  $\forall x \geq 0$ .)



# Analysis of MR SampleSort (cont'd)

Sorted sequence of  $N$  objects



## Analysis of MR SampleSort (cont'd)

### Proof of Lemma.

- ② (cont'd) So, there are  $K/(2 \ln N)$  segments and we know that, for each segment, the event “no splitter falls in the segment” occurs with probability  $\leq 1/N^2$ . Hence, the probability that any of these  $K/(2 \ln N)$  events occurs is  $\leq K/(N^2 2 \ln N) \leq 1/(2N)$  (union bound!). Therefore, with probability at least  $(1 - 1/(2N))$ , at least 1 splitter falls in each segment, which implies that each  $N_\ell$  cannot be larger than  $4(N/K) \ln N$ . Hence, we have that the second inequality stated in the lemma holds with probability at least  $(1 - 1/(2N))$ .

In conclusion, we have that the probability that at least one of the two inequalities does not hold, is at most  $2 \cdot 1/(2N) = 1/N$ , and the lemma follows.  $\square$

## Examples of theory questions

- In a MapReduce computation, each round transforms a set of key-value pairs into another set of key-value pairs, through a Map phase and a Reduce phase. Describe what the Map/Reduce phases do.
- What are the design goal that one should target when devising MapReduce algorithms?
- Briefly describe how to compute the product of an  $n \times n$  matrix by an  $n$ -vector in two rounds using  $o(n)$  local space

# Exercises

## Exercise 1

Design and analyze an efficient MR algorithm for the Category counting problem without assuming the input objects are provided with distinct keys in  $[0, N)$ . That is, the initial keys are values in some arbitrary domain.

## Exercise 2

Design and analyze an efficient MR algorithm for approximating the maximum pairwise distance assuming that a given upper bound  $M_L = O(f(n))$ , with  $f(n) = o(\sqrt{N})$  must be satisfied.

## Exercises (cont'd)

### Exercise 3

Exercise 2.3.1 of [LRU14] trying to come up with interesting tradeoffs between number of rounds and local space.

### Exercise 4

Generalize the matrix-vector and matrix multiplication algorithms to handle rectangular matrices

## References

- LRU14 J. Leskovec, A. Rajaraman and J. Ullman. Mining Massive Datasets. Cambridge University Press, 2014. Chapter 2 and Section 6.4
- DG08 J. Dean and A. Ghemawat. MapReduce: simplified data processing on large clusters. OSDI'04 and CACM 51,1:107113, 2008
- MU05 M. Mitzenmacher and E. Upfal. Proability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, 2005. (Chernoff bounds: Theorems 4.4 and 4.5)
- P+12 A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, E. Upfal: Space-round tradeoffs for MapReduce computations. ACM ICS'112.
- RU14 M. Riondato, E. Upfal: Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees. ACM Trans. on Knowledge Discovery from Data, 2014.