Computational Frameworks

MapReduce

Computational challenges in data mining

- Computation-intensive algorithms: e.g., optimization algorithms, graph algorithms
- Large inputs: e.g., web, social networks data.
  Observation: For very large inputs, superlinear complexities become unfeasible
- Parallel/distributed platforms are required
  - Specialized high-performance architectures are costly and become rapidly obsolete
  - Fault-tolerance becomes serious issue: low MTBF (Mean-Time Between Failures)
  - Effective parallel/distributed programming requires high skills

Example: Web index

About $50 \cdot 10^9$ web pages are indexed. Considering 20KB per page, this gives 1000TB of data. Just reading the whole index from secondary storage takes a substantial amount of time.

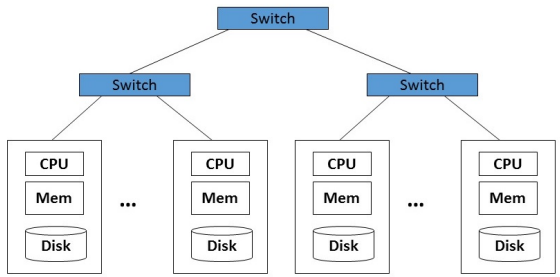
MapReduce

- 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

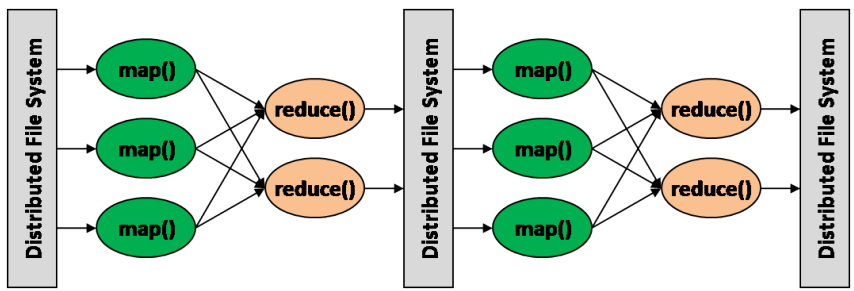
- Racks of 16-64 compute nodes (commodity hardware), connected (within each rack and among racks) by fast switches (e.g., 10 Gbps Ethernet)



- Distributed File System
  - Files divided into *chunks* (e.g., 64MB per chunk)
  - Each chunk replicated (e.g., 2x o 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 ≥ 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 ≥ 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 is assigned as input to a map task.
- The map task runs on a worker (a compute node) and applies the map function to each key-value pair of its assigned chunk, buffering the intermediate key-value pairs it produces in the local disk
- The intermediate key-values pairs are partitioned into Y buckets (key $k \rightarrow$ bucket hash(k) mod Y) and each bucket is assigned to a different reduce task. Note that each map task stores in its local disk pieces of various buckets, hence a bucket is initially spread among several disks.
- The reduce task runs on a worker (a compute node), sorts the key-value pairs in its 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)

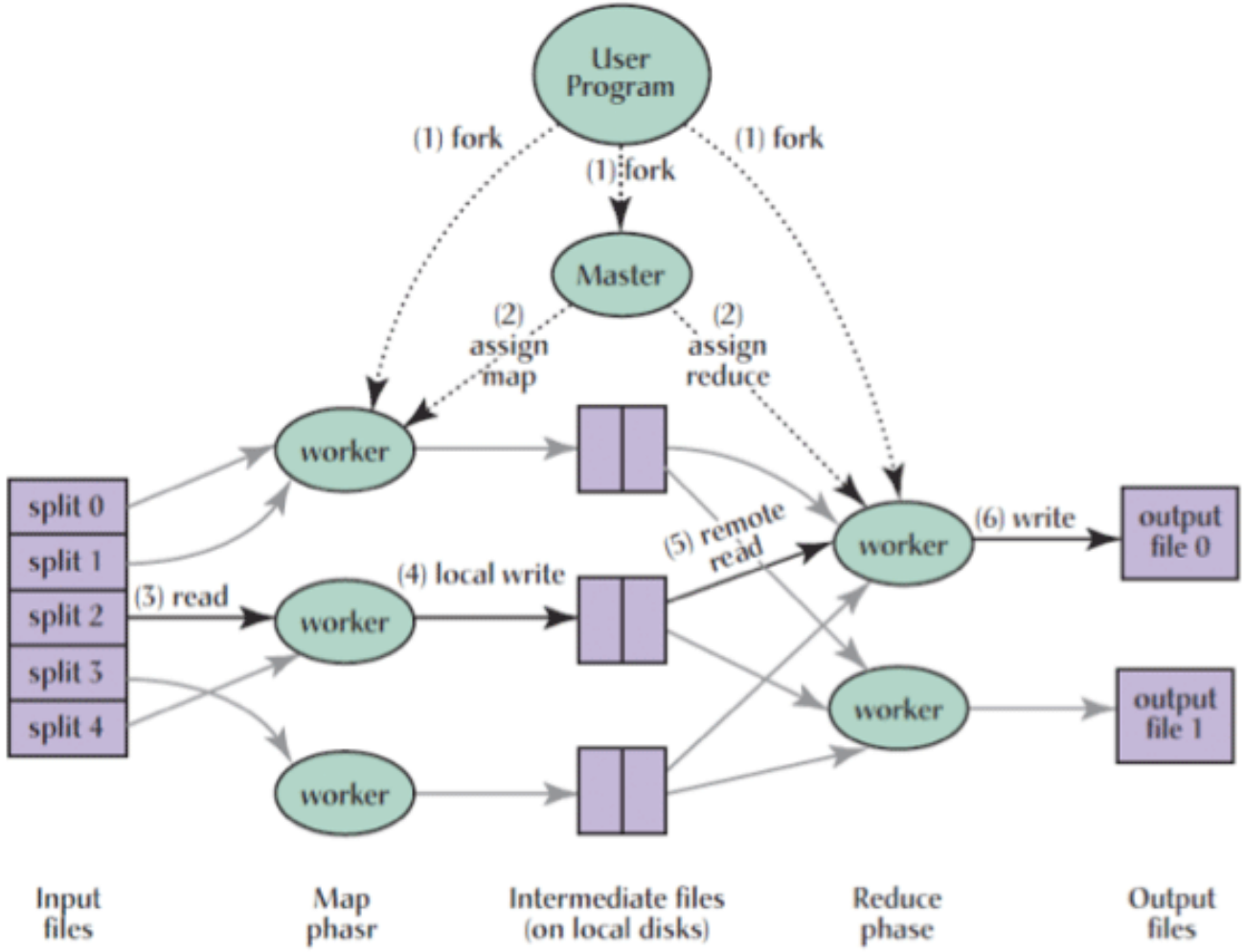
- The user program is forked into a master process and several worker processes. The master is in charge of assigning map and reduce tasks to the various workers, and to monitor their status (idle, in-progress, completed).
- Input and output files reside on a Distributed File System, while intermediate 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

MapReduce performance

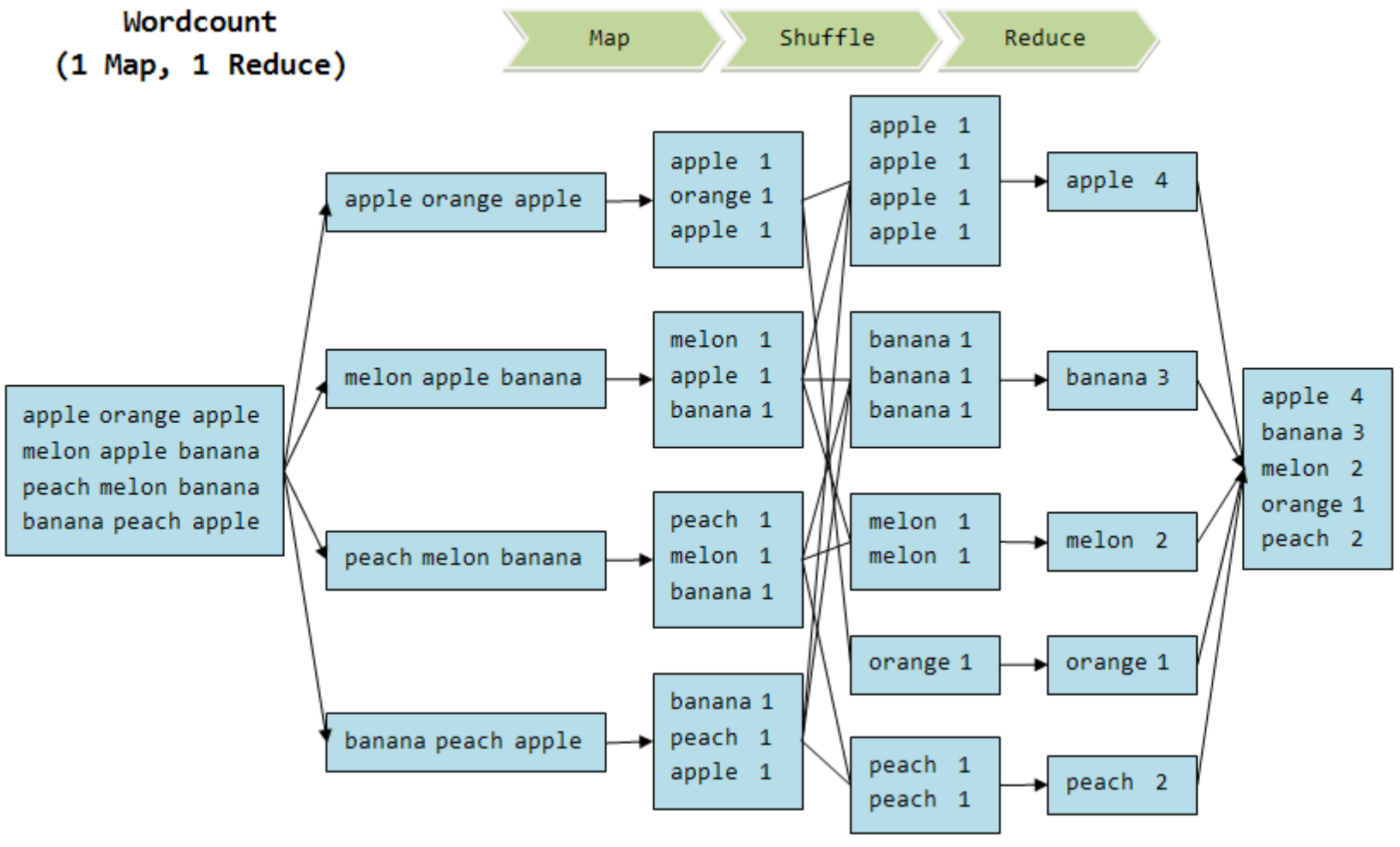
- Constraint: map and reduce functions must have polynomial complexities. In many practical scenarios, they are applied on to small subsets of the input and/or have linear complexity.
  $\Rightarrow$ running time of a round is dominated by the data shuffle
- Key performance indicators (see [P+12]):
  - Number of rounds R
  - Local space M_L: the maximum amount of space required by a map/reduce function for storing input and temporary data (does not the output of the function) Aggregate space M_A: the maximum space used in any round (aggregate space required by all map/reduce functions executed in the round)
- The complexity of a MapReduce algorithm is specified through R, which, in general, depends on the input size, on M_L, and on M_A. In general, the larger M_L and M_A, the smaller R.

Word Count

Example

- INPUT: collection of text documents $D_1, D_2, \ldots, D_k$ containing N words overall (counting repetitions).
- 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.
- MapReduce algorithm:
  - *Map phase*: consider each document as a key-value pair, where the key is the document name and the value is the document content. Given a document D_i, the map function applied to this document produces the set of 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*: group by key the intermediate pairs produced by the map phase, and for each word w sum the values (1's) of all intermediate pairs with key w, emitting $(w, c(w))$ in output.

Word Count (cont'd)



Word Count (cont'd)

Analysis

- R=1 round
- $M_L = O(N)$ (recall that N is the input size, namely, the total number of words in the documents). A bad case is when only one word occurs repeated N times over all documents, hence only one reducer is executed over a input of size $O(N)$.
- $M_A = \Theta(N)$

Remark: The following simple optimization reduces the space requirements. For each document 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 map function requires local space $O(\max_{i=1,k} N_i)$, and the reduce function requires local space $O(k)$. Hence, $M_L = O(\max_{i=1,k} N_i + k)$.

Observations

Theorem

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

The trivial solution implied by the above theorem is impractical for large inputs. For efficiency, algorithm design typically aims at

- Few rounds (e.g., $R = O(1)$)
- Sublinear local space (e.g., $M_L = O(|\text{input}|^{\epsilon})$, for some constant $\epsilon \in (0, 1)$)
- Linear aggregate space (i.e., $M_A = O(|\text{input}|)$), or only slightly superlinear

Observations (cont'd)

- In general, the domain of the keys (resp., values) in input to a map or reduce function is different from the domain of the keys (resp., values) produced by the function.
- 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 often omitted, when easily derivable) a MapReduce 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 MapReduce 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.

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

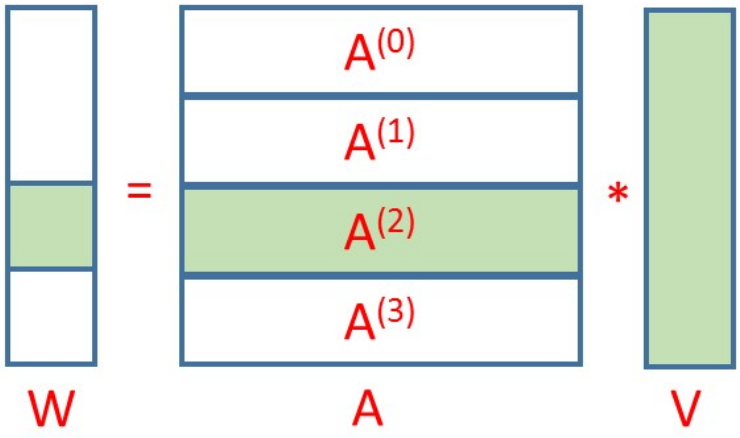
Trivial MapReduce algorithm:

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

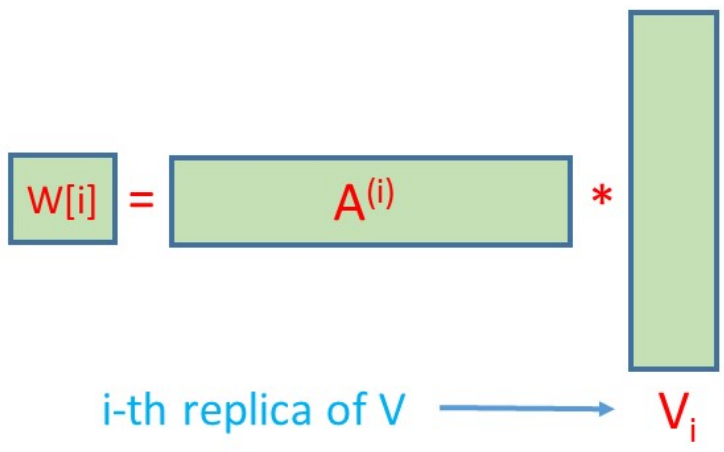
Analysis: $R = 1$ round; $M_L = \Theta(n)$; $M_A = \Theta(n^2)$ (i.e., sublinear local space and linear aggregate space).

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

Primitives: matrix-vector multiplication (cont'd)



Round 1
for each i in a
distinct reducer



Primitives: 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

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

Primitives: matrix-vector multiplication (cont'd)

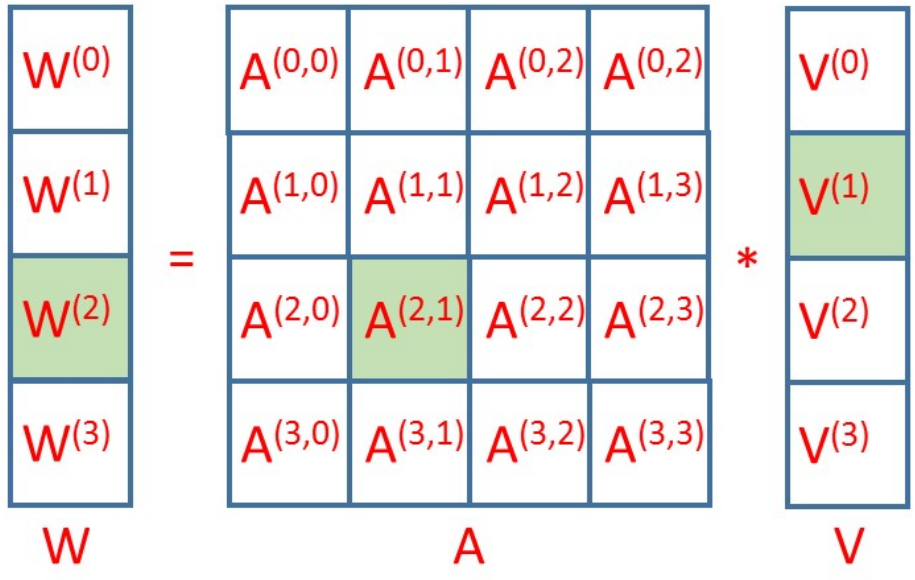
- Round 1:
  - *Map phase*: For every $0 \leq t < n/k$, create n/k replicas of $V^{(t)}$. Call these replicas $V_s^{(t)}$, with $0 \leq s < n/k$
  - *Reduce phase*: For every $0 \leq s, t < n/k$ in parallel, compute $W_t^{(s)} = A^{(s,t)} \cdot V_s^{(t)}$
- Round 2:
  - *Map phase*: identity
  - *Reduce phase*: For every $0 \leq s < n/k$ and $0 \leq t < n/k$, compute $W^{(s)} = \sum_{t=0}^{n/k-1} W_t^{(s)}$. The summation can be executed independently for each component of $W^{(s)}$

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

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

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

Primitives: matrix-vector multiplication (cont'd)



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 numberc*

Primitives: matrix multiplication

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

Output: $C = A \cdot B$

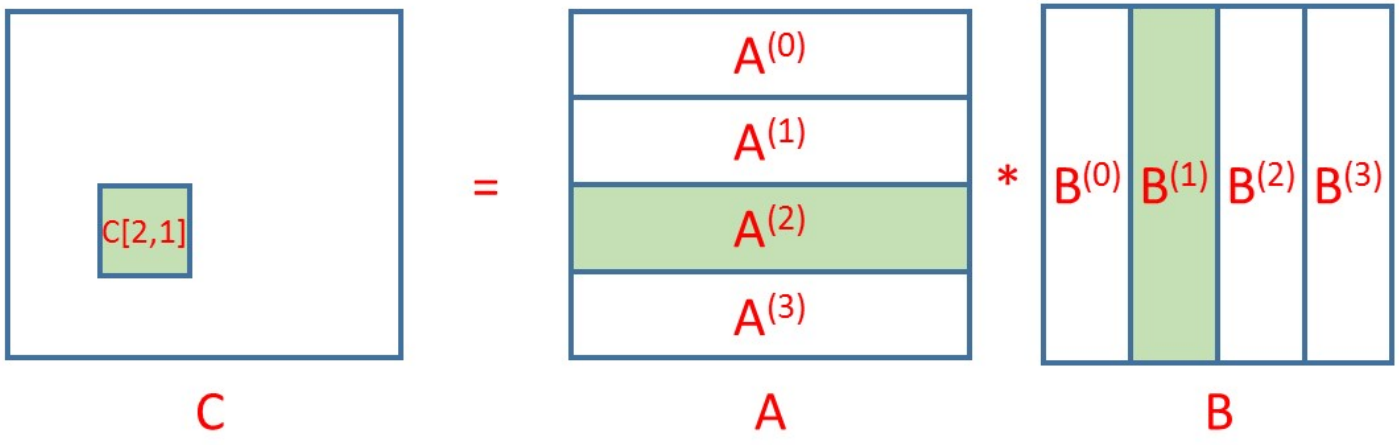
Trivial MapReduce algorithm:

- 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 $C[i, j] = A_j^{(i)} \cdot B_i^{(j)}$.

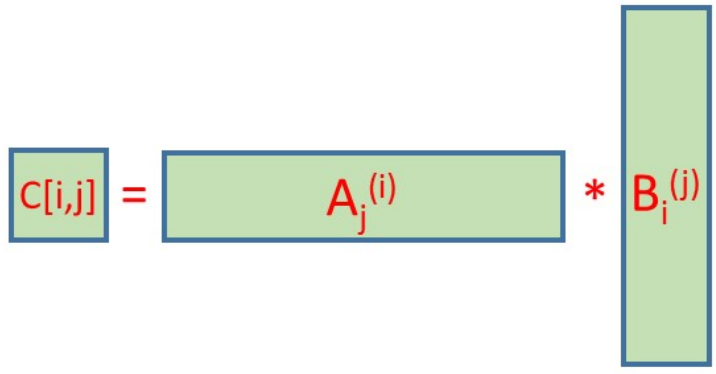
Analysis: $R = 1$ round; $M_L = \Theta(n)$; $M_A = \Theta(n^3)$ (i.e., sublinear local space but superlinear aggregate space).

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

Primitives: matrix multiplication (cont'd)



Round 1
for each i,j in a distinct reducer



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

Primitives: 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

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

Primitives: matrix multiplication (cont'd)

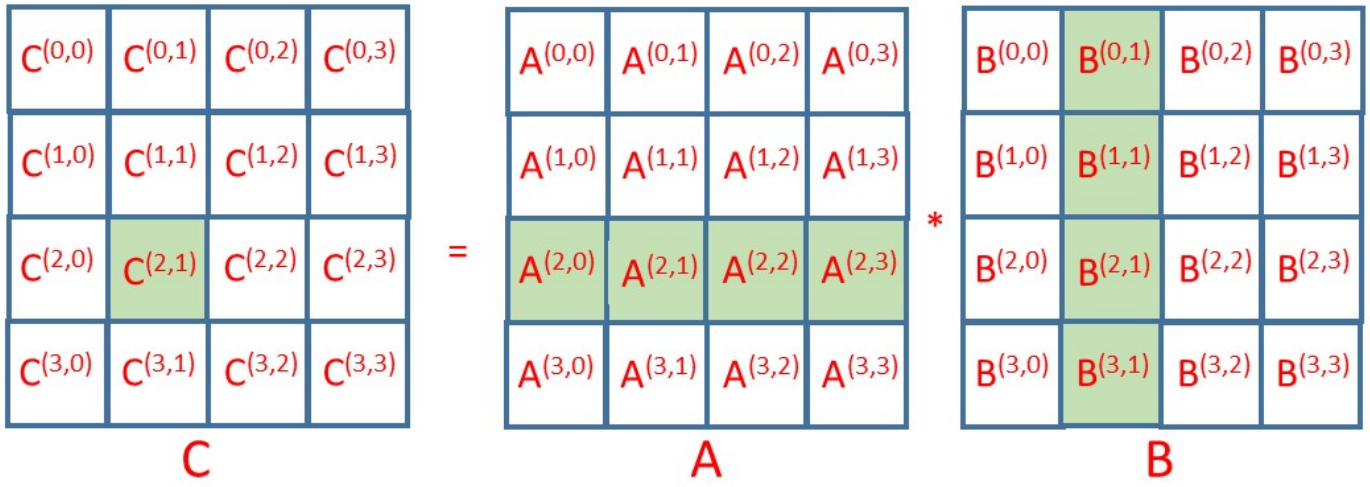
- Round 1:
  - *Map phase*: For every $0 \leq s, t < n/k$, create n/k replicas of $A^{(s,t)}$ and $B^{(s,t)}$. Call these replicas $A_i^{(s,t)}$ and $B_i^{(s,t)}$, with $0 \leq i < n/k$
  - *Reduce phase*: For every $0 \leq s, t, \ell < n/k$ in parallel, compute $C_\ell^{(s,t)} = A_t^{(s,\ell)} \cdot B_s^{(\ell,t)}$
- Round 2:
  - *Map phase*: identity
  - *Reduce phase*: For every $0 \leq s, t < n/k$, compute $C^{(s,t)} = \sum_{\ell=0}^{n/k-1} C_\ell^{(s,t)}$. The summation can be executed independently for each component of $C^{(s,t)}$

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

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

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

Primitives: matrix multiplication (cont'd)



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 the values of M_L and M_A are fixed and we must adapt the algorithm to comply with the given space constraints?

The presented algorithms can be generalized (see [P+12]) to require:

- Matrix-vector multiplication:

$$\begin{aligned} R &: O\left(\frac{\log n}{\log M_L}\right) \\ M_L &: \text{any fixed value} \\ M_A &: \Theta\left(n^2\right) \quad \text{(MINIMUM!)}. \end{aligned}$$

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

Observations (cont'd)

- Matrix multiplication:

$$\begin{array}{rcll} R & : & O\left(\frac{n^3}{M_A\sqrt{M_L}} + \frac{\log n}{\log M_L}\right) & \\ M_L & : & \text{any fixed value} & \\ M_A & : & \text{any fixed value} & \Omega\left(n^2\right) \end{array}$$

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

- 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.

The power of sampling

Primitives: 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 π is a permutation such that $s_{\pi(1)} \leq s_{\pi(2)} \leq \cdots \leq s_{\pi(N)}$.

MapReduce algorithm (*Sample Sort*):

- Let $K \in (\log N, N)$ be a suitable integral design parameter.
- Round 1:
  - *Map phase*: Select each object as a splitter with probability $p = K/N$, independently of the other objects, and replicate each splitter K times. Let $x_1 \leq x_2 \leq \cdots x_t$ be the selected splitters in sorted order, and define $x_0 = -\infty$ and $x_{t+1} = \infty$. Also, partition S arbitrarily into K subsets $S_0, S_1, \ldots S_{K-1}$. E.g., assign (i, s_i) to S_j with $j = i \mod K$.
  - *Reduce phase*: For $0 \leq j < K$ gather S_j and the splitters, and compute $S_j^{(i)} = \{s \in S_j \ : \ x_i < s \leq x_{i+1}\}$, for every $0 \leq i \leq t$

Primitives: sorting (cont'd)

- Assume the output of Round 1 consists of a key-value pair (i, s) for each object $s \in S_j^{(i)}$ (index j is irrelevant).
- Round 2:
  - *Map phase*: identity
  - *Reduce phase*: For every $0 \leq i \leq t$ gather $S^{(i)} = \{s \in S \ : \ x_i < s \leq x_{i+1}\}$ and compute $N_i = |S^{(i)}|$.
- Round 3:
  - *Map phase*: Replicate the vector $(N_0, N_1, \ldots, N_t)$ $t+1$ times.
  - *Reduce phase*: For every $0 \leq i \leq t$: gather $S^{(i)}$ and vector $(N_0, N_1, \ldots, N_t)$; sort $S^{(i)}$; and compute the final output pairs for the objects in $S^{(i)}$ (ranks starts from $1 + \sum_{\ell=0}^{i-1} N_\ell$.)

Example

$$N = 32$$

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

Round 1: Determine partition and splitters

$$\begin{aligned} S_0 &= \mathbf{16}, 32, 1, 15, 14, 7, 28, 20 \\ S_1 &= 12, 3, \mathbf{29}, 17, 11, 10, 8, 2 \\ S_2 &= 25, \mathbf{21}, 13, 5, 19, 23, 30, 26 \\ S_3 &= 31, 22, \mathbf{9}, 6, 27, 24, \mathbf{4}, 18 \end{aligned}$$

The $t = 5$ splitters (highlighted in blue) are: 16, 29, 21, 9, 4

Example (cont'd)

Round 1 (cont'd): Compute the $S_j^{(i)}$'s

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

Round 2 Gather the $S^{(i)}$'s and compute the N_i's

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

Example (cont'd)

Round 3: Compute the final output

- $S^{(0)}$ in sorted order from rank 1
- $S^{(1)}$ in sorted order from rank $N_0 + 1 = 5$
- $S^{(2)}$ in sorted order from rank $N_0 + N_1 + 1 = 10$
- $S^{(3)}$ in sorted order from rank $N_0 + N_1 + N_2 + 1 = 17$
- $S^{(4)}$ in sorted order from rank $N_0 + N_1 + N_2 + N_3 + 1 = 22$
- $S^{(5)}$ in sorted order from rank $N_0 + \cdots + N_4 + 1 = 30$

Analysis of SampleSort

- Number of rounds: $R = 3$
- Local Space M_L:
  - Round 1: $\Theta(t + N/K)$, since each reducer must store the entire set of splitters and one subset S_j
  - Round 2: $\Theta(\max\{N_i\ ;\ 0 \leq i \leq t\})$ since each reducer must gather one $S^{(i)}$.
  - Round 3: $\Theta(t + \max\{N_i\ ;\ 0 \leq i \leq t\})$, since each reducer must store all N_i's and one $S^{(i)}$.

  $\Rightarrow$ overall $M_L = \Theta(t + N/K + \max\{N_i\ ;\ 0 \leq i \leq t\})$
- Aggregate Space M_A: $O\left(N + t \cdot K + t^2)\right)$, since in Round 1 each splitter is replicated K times, and in Round 3 the vector $(N_0, N_1, \ldots, N_t)$ is replicated $t + 1$ times, while the objects are never replicated

Analysis of SampleSort (cont'd)

Lemma

For a suitable constant $c > 1$, the following two inequalities hold with high probability (i.e., probability at least $1 - 1/N$):

1. $t \leq cK$, *and*
2. $\max\{N_i \,;\, 0 \leq i \leq t\} \leq c(N/K)\log N$.

Proof.

Deferred. □

Theorem

By setting $K = \sqrt{N}$, the above algorithm runs in 3 rounds, and requires local space $M_L = O\left(\sqrt{N}\log N\right)$ and aggregate space $M_A = O(N)$, with high probability.

Proof.

Immediate from lemma. □

Analysis of SampleSort (cont'd)

Chernoff bound (see [MU05])

Let $X_1, X_2, \ldots, X_n$ 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 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

$$\begin{aligned} \Pr(X \geq (1+\delta_1)\mu) &\leq 2^{-(1+\delta_1)\mu} \\ \Pr(X \leq (1-\delta_2)\mu) &\leq 2^{-\mu\delta_2^2/2} \end{aligned}$$

Analysis of 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 $E[t] = K > \log N$. By choosing c large enough, the Chernoff bound shows that $t > cK$ with probability at most $1/(2N)$. For example, choose $c \geq 6$ and apply the Chernoff bound with $\delta_1 = 5$.
2. View the *sorted* sequence of objects as divided into $K/(\alpha \log N)$ contiguous segments of length $N' = \alpha(N/K) \log N$ each, for a suitably large constant $\alpha > 0$, and consider one such segment. The number of splitters that fall in the segment is a Binomial$(N', K/N)$ random variable, whose mean is $\alpha \log N$. By using Chernoff bound we can show that the probability that no splitter falls in the segment is at most $1/N^2$. For example, choose $\alpha = 16$ and apply the Chernoff bound with $\delta_2 = 1/2$.

Analysis of SampleSort (cont'd)

Proof of Lemma.

2. (cont'd) So, there are $K/(\alpha \log N)$ 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 at least one of these $K/(\alpha \log N)$ events occur is $\leq K/(N^2 \alpha \log N) \leq 1/(2N)$ (union bound!). Therefore, with probability at least $(1 - 1/(2N)))$, at least 1 splitter falls in each segment, which implies that each N_i cannot be larger than $2\alpha(N/K) \log N$. Hence, by choosing $c \geq 2\alpha$ we have that the second inequality stated in the lemma holds with probability at least $(1 - 1/(2N)))$.

In conclusion, by setting $c = \max\{6, 2\alpha\} = 32$ 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. □

Primitives: frequent itemsets

Input: Set T of N transactions over a set I of items, and support threshold minsup. Each transaction represented as a pair (i, t_i), where i is the TID ($0 \leq i < N$), and $t_i \subseteq I$

Output: Set of frequent itemsets w.r.t. T and minsup, and their supports.

MapReduce algorithm (based on SON algorithm [VLDB'95]):

- Let K be an integral design parameter, with $1 < K < N$.
- Round 1:
  - *Map phase*: Partition T arbitrarily into K subsets $T_0, T_1, \ldots T_{K-1}$ of $O(N/K)$ transactions each. E.g., assign transaction (i, t_i) to T_j with $j = i \mod K$.
  - *Reduce phase*: For $0 \leq j \leq K-1$ gather T_j and minsup, and compute the set of frequent itemsets w.r.t. T_j and minsup. Each such itemset X will be represented by a pair (X, null)

Primitives: frequent itemsets (cont'd)

- Round 2:
  - *Map phase*: identity
  - *Reduce phase*: Eliminate duplicate pairs from the output of Round 1. Let Φ be the resulting set of pairs
- Round 3:
  - *Map phase*: Replicate K times each pair of Φ
  - *Reduce phase*: For every $0 \leq j < K$ gather Φ and T_j and, for each $(X, \text{null}) \in \Phi$, compute a pair $(X, s(X, j))$ with $s(X, j) = \text{Supp}_{T_j}(X)$.
- Round 4:
  - *Map phase*: identity
  - *Reduce phase*: For each $X \in \Phi$ compute $s(X) = (1/N) \sum_{j=0}^{K-1} (|T_j| \cdot s(X, j))$, and output $(X, s(X))$ if $s(X) \geq \text{minsup}$.

Analysis of SON algorithm

- Correctness: it follows from the fact that *each itemset frequent in T must be frequent in some T_j*.
- Number of rounds: 4.
- Space requirements: Assume that each transaction has length $\Theta(1)$ (the analysis of the general case is left as an exercise). Let $\mu = \Omega(N/K)$ be the maximum local space used in Round 1, and let $|\Phi|$ denote the number of itemsets in Φ, The following bounds are easily established
  - Local space $M_L = O(K + \mu + |\Phi|)$
  - Aggregate space $M_A = O(N + K \cdot (\mu + |\Phi|))$

Observations:

- The values μ and $|\Phi|$ depend on several factors: the partition of T, the support threshold, the algorithm used to extract frequent itemsets from each T_j. If A-Priori is used, we know that $\mu = O(|I| + |\Phi|^2)$
- In any case, $\Omega\left(\sqrt{N}\right)$ local space is needed, which can be a lot

Primitives: frequent itemsets (cont'd)

Do we really need to process the entire dataset?

No, if we are happy with some

approximate set of frequent itemsets

(but quality of approximation under control)

What follows is based on [RU14]

Primitives: frequent itemsets (cont'd)

Definition (Approximate frequent itemsets)

Let T be a dataset of transactions over the set of items I and minsup $\in (0, 1]$ a support threshold. Let also $\epsilon > 0$ be a suitable parameter. A set C of pairs (X, s_X), with $X \subseteq I$ and $s_x \in (0, 1]$, is an ϵ-approximation of the set of frequent itemsets and their supports if the following conditions hold:

1. For each $X \in F_{T,\text{minsup}}$ there exists a pair $(X, s_X) \in C$
2. For each $(X, s_X) \in C$,
   - $\text{Supp}(X) \geq \text{minsup} - \epsilon$
   - $|\text{Supp}(X) - s_X| \leq \epsilon$,

where $F_{T,\text{minsup}}$ is the true set of frequent itemsets w.r.t. T and minsup.

Primitives: frequent itemsets (cont'd)

Observations

- Condition (1) ensures that the approximate set C comprises all true frequent itemsets
- Condition (2) ensures that: (a) C does not contain itemsets of very low support; and (b) for each itemset X such that $(X, s_X) \in C$, s_X is a good estimate of its support.

Primitives: frequent itemsets (cont'd)

Simple sampling-based algorithm

Let T be a dataset of N transactions over I, and $\text{minsup} \in (0, 1]$ a support threshold. Let also $\theta(\text{minsup}) < \text{minsup}$ be a suitably lower support threshold

- Let $S \subseteq T$ be a sample drawn at random with replacement (uniform probability)
- Return the set of pairs

$$C = \left\{ (X, s_x = \text{Supp}_S(X)) \ : \ X \in F_{S,\theta(\text{minsup})} \right\},$$

where $F_{S,\theta(\text{minsup})}$ is the set of frequent itemsets w.r.t. S and $\theta(\text{minsup})$.

How well does C approximate the true frequent itemsets and their supports?

Primitives: frequent itemsets (cont'd)

Theorem (Riondato-Upfal)

Let h be the maximum transaction length and let ϵ, δ be suitable design parameters in $(0, 1)$. There is a constant $c > 0$ such that if

$$\theta(\mathit{minsup}) = \mathit{minsup} - \frac{\epsilon}{2} \quad \mathit{AND} \quad |S| = \frac{4c}{\epsilon^2}\left(h + \log\frac{1}{\delta}\right)$$

then with probability at least $1 - \delta$ the set C returned by the algorithm is an ϵ-approximation of the set of frequent itemsets and their supports.

The proof of the theorem requires the notion of VC-dimension.

VC-dimension [Vapnik,Chervonenkis 1971]

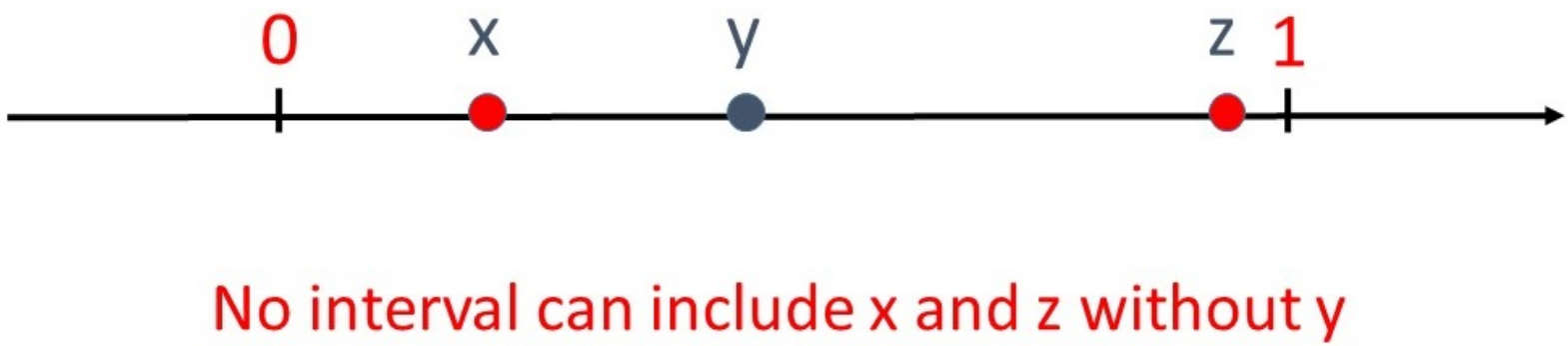
Powerful notion in statistics and learning theory

Definition

A Range Space is a pair (D, R) where D is a finite/infinite set (*points*) and R is finite/infinite family of subsets of D (*ranges*). Given $A \subset D$, we say that A is shattered by R if the set $\{r \cap A \ : \ r \in R\}$ contains all possible subsets of A. The VC-dimension of the range space is the cardinality of the largest $A \subset D$ which is shattered by R.

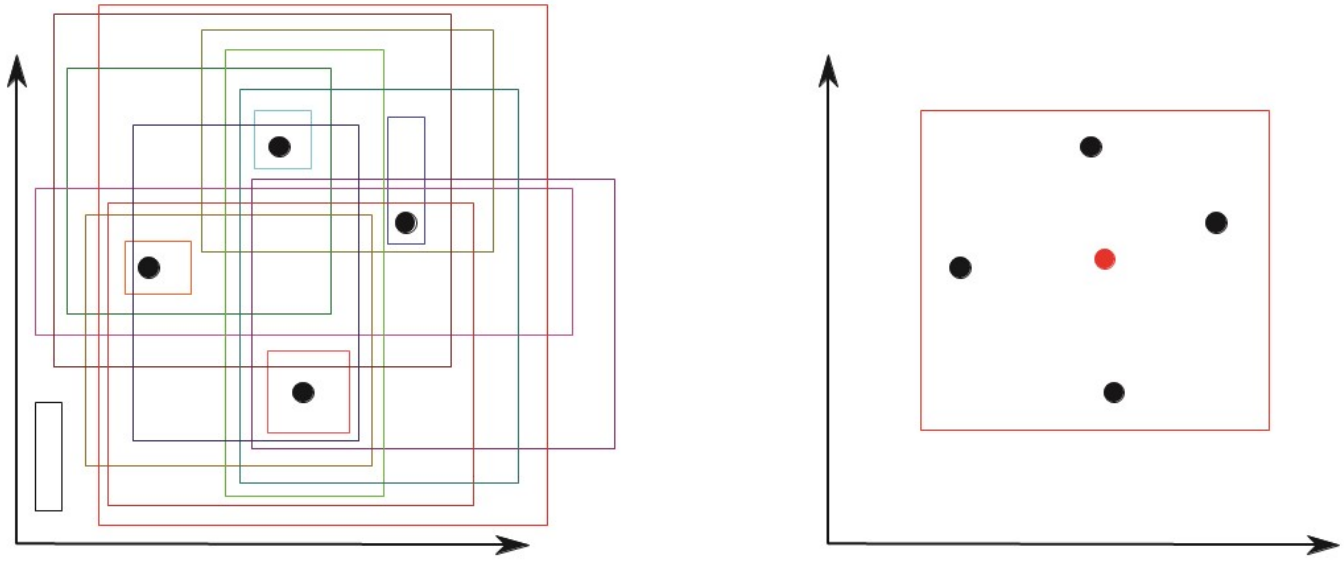
VC-dimension: examples

Let $D = [0, 1]$ and let R be the family of intervals $[a, b] \subseteq [0, 1]$. It is easy to see that the VC-dimension of (D, R) is ≥ 2. Consider any 3 points $0 \leq x < y < z \leq 1$. The following picture show that it is < 3, hence it must be equal to 2.



VC-dimension: examples (cont'd)

Let $D = \Re^2$ and let R be the family of axis-aligned rectangles.



There exist 4 points shattered by R (left), but no set of 5 points can be shattered by R (right)

$\Rightarrow$ the VC-dimension of (D, R) is 4.

Primitives: frequent itemsets (cont'd)

Lemma (Sampling Lemma)

Let (D, R) be a range space with VC-dimension v with D finite and let $\epsilon_1, \delta \in (0, 1)$ be two parameters. For a suitable constant $c > 0$, we have that given a random sample $S \subseteq D$ (drawn from D with replacement and uniform probability) of size

$$m \geq \min\left\{|D|, \frac{c}{\epsilon_1^2}\left(v + \log\frac{1}{\delta}\right)\right\}$$

with probability at least $1 - \delta$, we have that for any $r \in R$

$$\left|\frac{|r|}{|D|} - \frac{|S \cap r|}{|S|}\right| \leq \epsilon_1$$

We will not prove the lemma. See [RU14] for pointers to proof.

Primitives: frequent itemsets (cont'd)

A dataset T of transactions over I can be seen as a range space (D, R):

- $D = T$
- $R = \{T_X \ : \ X \subseteq I \wedge X \neq \emptyset\}$, where T_X is the set of transactions that contain X.

It can be shown that the VC-dimension of (D, R) is $\leq h$, where h is the maximum transaction length.

Primitives: frequent itemsets (cont'd)

Proof of Theorem (Riondato-Upfal).

Regard T as a range space (D, R) of VC-dimension h, as explained before. The Sampling Lemma with $\epsilon_1 = \epsilon/2$ shows that with probability $\geq 1 - \delta$ for each itemset X it holds $|\text{Supp}_T(X) - \text{Supp}_S(X)| \leq \epsilon/2$. Assume that this is the case. Therefore:

- For each frequent itemset $X \in F_{T,\text{minsup}}$

$$\text{Supp}_S(X) \geq \text{Supp}_T(X) - \epsilon/2 \geq \text{minsup} - \epsilon/2 = \theta(\text{minsup}),$$

  hence, the pair $(X, s_X = \text{Supp}_S(X))$ is returned by the algorithm;

- For each pair $(X, s_X = \text{Supp}_S(X))$ returned by the algorithm

$$\text{Supp}_T(X) \geq s_X - \epsilon/2 \geq \theta(\text{minsup}) - \epsilon/2 \geq \text{minsup} - \epsilon$$

  and $|\text{Supp}_T(X) - \text{Supp}_S(X)| \leq \epsilon/2 < \epsilon$.

Hence, the output of the algorithm is an ϵ-approximation of the true frequent itemsets and their supports. □

Observations

- The size of the sample is independent of the support threshold minsup and of the number N of transactions. It only depends on the approximation guarantee embodied in the parameters ϵ, δ, and on the max transaction length h, which is often quite low.
- There are bounds on the VC-dimension of the range space tighter than h.
- The sample-based algorithm yields a 2-round MapReduce algorithm: in first round the sample of suitable size is extracted (see exercise 5); in the second round the frequent itemsets are extracted from the sample with one reducer.
- The sampling approach can be boosted by extracting frequent itemsets from several smaller samples, returning only itemsets that are frequent in a majority of the samples. In this fashion we may end up doing globally more work but in less time because the samples can be mined in parallel.

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 is the goal one should target when devising a MapReduce solution for a given computational problem?
- 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
- Let T be a dataset of N transactions, partitioned into K subsets $T_0, T_1, \ldots, T_{K-1}$. For a given support threshold minsup, show that any frequent itemset w.r.t. T and minsup must be frequent w.r.t. some T_i and minsup.

Exercises

Exercise 1

Let T be a huge set of web pages gathered by a crawler. Develop an efficient MapReduce algorithm to create an inverted index for T which associates each word w to the list of URLs of the pages containing w.

Exercise 2

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

Exercise 3

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

Exercises (cont'd)

Exercise 4

Generalize the analysis of the space requirements of SON algorithm to the case when transactions have arbitrary length. Is there a better way to initially partition T?

Exercise 5

Consider a dataset T of N transactions over I given in input as in algorithm SON. Show how to draw a sample S of K transactions from T, uniformly at random with replacement, in one MapReduce round. How much local space is needed by your method?

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.