Exercises on MapReduce

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

Solution. The input of the problem is a set S of N objects represented by pairs $(o_i, \gamma_i))$, for $0 \leq i < N$, where o_i is the i-th object, and γ_i its category. Let us assume for simplicity that the objects, which belong to some arbitrary domain, are the keys for the pairs. We want to compute the set of pairs $(\gamma, c(\gamma))$ where γ is a category labeling some object of S and $c(\gamma)$ is the number of objects of S labeled with γ. In order to maintain the local space requirements under control, we can solve the problem in two rounds: in the first round we partition at random the input pairs into $\sqrt{N}$ subsets and compute partial counts within each subset independently; in the second round we aggregate the partial counts. The details of the algorithm are below.

Round 1

- *Map phase*: map each pair (o_i, γ_i) into the intermediate pair $(x, (o_i, \gamma_i))$, where x (the key of the pair) is a random value in $[0, \sqrt{N})$.
- *Reduce phase*: For each key $x \in [0, \sqrt{N})$ independently, gather the set (say $S^{(x)}$) of all intermediate pairs with key x and, for each category γ labeling some object in $S^{(x)}$, produce the pair $(\gamma, c_x(\gamma))$, where $c_x(\gamma)$ is the number of objects of $S^{(x)}$ labeled with γ.

Round 2

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

The analysis is similar to the one presented in class for Improved Word Count 2. Let m_x be the number of intermediate pairs with key x produced by the Map phase of Round 1, and let $m = \max_x m_x$. It is easy to see that the above 2-round algorithm requires local space $M_L = O\left(m + \sqrt{N}\right)$ and aggregate space $M_A = O\left(N\right)$. The theorem stated in Slide 25 of the slides on MapReduce shows that $m = O\left(\sqrt{N}\right)$ with high probability, namely probability $\geq 1 - 1/N^5$. Hence, $M_L = O\left(\sqrt{N}\right)$ with probability at least $1 - 1/N^5$. $\square$

Exercise. Design an MR algorithm which solves (exactly) the Maximum pairwise distance problem for a set S of N points from a metric space (M, d). The algorithm must run in $R = O(1)$ rounds and require local space $M_L = O\left(\sqrt{N}\right)$ and aggregate space $M_A = O(N^2)$.

Solution. Let the set S be represented by the set of pairs $\{(i, x_i) \ : \ 0 \le i < N\}$, where x_i denotes the i-th point. We need to compute the value

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

The idea is to first compute all distances between the $\binom{N}{2}$ possible pairs and then compute the maximum in several rounds processing distances in groups of size $\sqrt{N}$. The details of the algorithm are below.

Round 1

- *Map phase*: map each pair (i, x_i) into the following $N-1$ intermediate pairs: $((i,j), x_i)$, for $i < j < N$, and $((j,i), x_i)$, for $0 \le j < i$.
- *Reduce phase*: For every (i,j) with $0 \le i < j < N$ independently, gather the two intermediate pairs $((i,j), x_i)$ and $((i,j), x_j)$ and produce the pair $(k_{ij}, d(x_i, x_j))$, with $k_{ij} = i \cdot N + j$. Note that the k_{ij}'s are all distinct and in the range $[0, N^2)$.

Round 2

- *Map phase*: map each pair (k, d) into the intermediate pair $(k' = k \bmod N^{3/2}, d)$ and note that $k' \in [0, N^{3/2})$.
- *Reduce phase*: for each $k' \in [0, N^{3/2})$ independently, gather the $\le \sqrt{N}$ intermediate pairs $(k'd)$ and produce the pair $(k'd')$ where d' is the maximum of the values of all such intermediate pairs.

Round 3

- *Map phase*: map each pair (k, d) into the intermediate pair $(k' = k \bmod N, d)$ and note that $k' \in [0, N)$.
- *Reduce phase*: for each $k' \in [0, N)$ independently, gather the $\le \sqrt{N}$ intermediate pairs $(k'd)$ and produce the pair $(k'd')$ where d' is the maximum of the values of all such intermediate pairs.

Round 4

- *Map phase*: map each pair (k, d) into the pair $(k' = k \bmod \sqrt{N}, d)$ and note that $k' \in [0, \sqrt{N})$.

- *Reduce phase*: for each $k' \in [0, \sqrt{N})$ independently, gather the $\leq \sqrt{N}$ intermediate pairs $(k'd)$ and produce the pair $(0, d')$ where d' is the maximum of the values of all such intermediate pairs.

Round 5

- *Map phase*: identity mapping.
- *Reduce phase*: gather the $\leq \sqrt{N}$ pairs $(0, d)$ and compute the value $d_{\max}$ as the maximum of the values of all such pairs.

It is easy to see that each of the 5 rounds requires local space $M_L = O\left(\sqrt{N}\right)$, while the required aggregate space is $M_A = O\left(N^2\right)$ since in Round 1 the distances between all pairs of points are computed and stored. $\square$

Exercise. Design an MR algorithm which computes an approximate solution to the Maximum pairwise distance problem for a set S of N points from a metric space (M, d) assuming that a given upper bound $M_L = O\left(f(N)\right)$, with $f(N) = o(\sqrt{N})$ must be satisfied.

Solution. Let the set S be represented as in the previous exercise. We can modify the algorithm presented in Slide 33 of the slides on MapReduce which approximates the maximum pairwise distance $d_{\max}$ through the maximum distance $d_{\max}(0)$ between x_0 and all other points. In particular, in Round 1 the N points are subdivided into $N/f(N)$ groups of size $f(N)$ each (for simplicity we assume that $f(N)$ divides N) and for each group the maximum distance from x_0 to a point in the group is computed. Then, the maximum of the $N/f(N)$ distances derived in Round 1 can be computed in several rounds by processing the distances in groups of size $f(N)$. The computation is structured as a tree of ariety $f(N)$, whose leaves are the $N/f(N)$ distances derived in Round 1. Specifically, Round i, with $i \geq 2$, starts with a set of $N/(f(N))^{i-1}$ distances, subdivides them into $N/(f(N))^i$ groups of size $f(N)$ each, and computes the maximum in each group. The algorithm terminates at the first Round i such that $N/(f(N))^{i-1} < f(N)$. At this point all distances are gathered and the final maximum is computed. The detailed specification of the algorithm is trivial an it is left as an exercise (assume for simplicity that N is a power of $f(N)$). It is easy to see that the algorithm requires local space $M_L = O\left(f(N)\right)$ and linear aggregate space. $\square$

Exercise. (Exercise 2.3.1.(b) of [LRU14]) Design an efficient MapReduce algorithm to compute the average of a set S of N integers, coming up with interesting round-space tradeoffs.

Solution. Suppose that S is provided in input as a set of key-value pairs (i, x_i) where x_i is an arbitrary integer and i, with $0 \leq i < N$, is the key of the pair. While the trivial 1-round algorithm uses $\Theta\left(N\right)$ local space, the space requirements can be substantially lowered using the following simple 2-round strategy.

Round 1

- *Map phase*: map each pair (i, x_i) into the intermediate pair $(i \bmod \sqrt{N}, x_i)$ (for simplicity, assume that $\sqrt{N}$ is an integer).
- *Reduce phase*: for each key k independently, with $0 \le k < \sqrt{N}$, gather the set S_k of all intermediate pairs (k, x) and produce the pair $(0, \text{avg}_k))$, where $\text{avg}_k = (1/N) \sum_{(k,x) \in S_k} x$.

Round 2

- *Map phase*: identity mapping.
- *Reduce phase*: gather all pairs $(0, \text{avg}_k)$ and return $\text{avg} = \sum_{(0,\text{avg}_k)} \text{avg}_k$.

It is immediate to see that the algorithm requires $M_L = \Theta\left(\sqrt{N}\right)$ and aggregate space $M_A = \Theta(N)$.

Observation. We can modify the above algorithm so that it uses local space M_L, for any fixed value M_L, as follows. In the first round, the input set S is partitioned into N/M_L subsets of size M_L each, and the contribution each subset to the average is computed. Then, the final average is computed in $O\left(\log_{M_L}(N/M_L)\right) = O\left(\log(N/M_L)/\log(M_L)\right)$ additional rounds, using a summation tree of ariety M_L whose leaves are the N/M_L contributions to be summed up. The number of rounds remains constant as long as $M_L = \Omega\left(N^{\epsilon}\right)$ for some constant $\epsilon > 0$. $\square$

Exercise. (Exercise 2.3.1.(d) of [LRU14]) Design an efficient MapReduce algorithm to compute the number of distinct integers in a set S of N integers, coming up with interesting round-space tradeoffs.

Solution. As before, we assume that S is provided in input as a set of key-value pairs (i, x_i) where x_i is an arbitrary integer and i, with $0 \le i < N$, is the key of the pair. We present a straightforward 2-round algorithm and a more space-efficient 4-round algorithm. The 2-round algorithm does a simple removal of duplicates as follows:

Round 1

- *Map phase*: map each pair (i, x_i) into the intermediate pair (x_i, i), that is, x_i becomes the key.
- *Reduce phase*: for each key x independently, gather the set S_x of all intermediate pairs (x, j) and produce the unique pair $(0, x)$.

Round 2

- *Map phase*: identity mapping.
- *Reduce phase*: gather the set S_0 of all pairs $(0, x)$ return $\text{count} = |S_0|$.

Let K_1 be the maximum number of occurrences of the same integer and K_2 the number of distinct integers in the input set S. Observe that $K_1 \geq N/K_2$. It is immediate to see that the algorithm requires $M_L = \Theta(K_1 + K_2)$ and aggregate space $M_A = \Theta(N)$. Note that, since $K_1 \geq N/K_2$, we have that, in general, $M_L = \Omega\left(\sqrt{N}\right)$, but it may become much larger than $\sqrt{N}$.

The following is an alternative strategy that reduces the local space requirements to $O\left(\sqrt{N}\right)$.

Round 1

- *Map phase*: map each pair (i, x_i) into the intermediate pair $(i \bmod \sqrt{N}, x_i)$.
- *Reduce phase*: for each key $j \in [0, \sqrt{N})$ independently, gather the set S_j of all intermediate pairs (j, x) and for each pair (j, x) remove all duplicates except one.

Note that after Round 1, for every integer x there are at most $\sqrt{N}$ pairs (j, x), one for each value of $j \in [0, \sqrt{N})$.

Round 2

- *Map phase*: map each pair (j, x) into the intermediate pair (x, j), that is swap key and value.
- *Reduce phase*: for each x independently, gather the set S_x of all intermediate pairs (x, j) and select, arbitrarily, only one such pair.

Note that after Round 2, for every integer x only one pair survives, and for each value of $j \in [0, \sqrt{N})$ there are at most $\sqrt{N}$ pairs with value j.

Round 3

- *Map phase*: map each pair (x, j) into the intermediate (j, x), that is swap again key and value.
- *Reduce phase*: for each $j \in [0, \sqrt{N})$ independently, gather all intermediate pairs (j, x) and produce the pair $(0, c_j)$, where c_j is the number of such pairs. (In fact, c_j is a partial count of the distinct integers.

Round 4

- *Map phase*: identity map.
- *Reduce phase*: gather all pairs $(0, c_j)$ with $j \in [0, \sqrt{N})$ and return the number of distinct integers as $\sum_{j=0}^{\sqrt{N}} c_j$.

It is easy to see that the above 4-round algorithm requires local space $M_L = \Theta\left(\sqrt{N}\right)$ and aggregate space $M_A = \Theta(N)$. As an example, apply the algorithm to the following input set

$$\begin{aligned} S \;=\; & \{(0,21),(1,120),(2,21),(3,76),(4,76),(5,76),(6,101),(7,120), \\ & (8,21),(9,120),(10,120),(11,76),(12,76),(13,76),(14,560),(15,21)\}. \end{aligned}$$

□

Exercise. Specify input, output and intermediate key-value pairs for the space-efficient 2-round MR algorithm for matrix-vector multiplication presented in Slides 37-39 of the slides on MapReduce.

Solution. Let A and V denote the input $n \times n$-matrix and n-vector, and let $W = A \cdot V$. We assume that A is initially represented through n^2 pairs $((i,j), A[i,j])$, with $0 \le i,j < n$, where (i,j) is the key, and that V (resp., W) is represented through n pairs $(i, V[i])$ (resp., $(i, W[i])$), with $0 \le i < n$, where i is the key.

Round 1

- *Map phase*: map each input pair $((i,j), A[i,j])$ into the intermediate pair $((i,s),((i,j),A[i,j]))$, with $s = \lfloor j/k \rfloor$, where (i,s) is the key. Thus, segment $A^{(i,s)}$ will be represented by all such pairs whose key is (i,s). Also, input pair $(i,V[i])$ is mapped into the n/k intermediate pairs $((i,s),(i,V[i]))$ with $0 \le s < n/k$, where (i,s) is the key. Thus, the i-th replica of segment $V^{(s)}$ (denoted by $V_i^{(s)}$) will be represented by all such pairs whose key is (i,s).
- *Reduce phase*: for each $0 \le i < n$ and $0 \le s < n/k$ independently, gather the intermediate pairs representing $A^{(i,s)}$ and $V_i^{(s)}$ and produce the pair $(i, W_s[i])$, where $W_s[i] = A^{(i,s)} \cdot V_i^{(s)}$ is the s-th contribution to the final value $W[i]$.

Round 2

- *Map phase*: identity mapping.
- *Reduce phase*: for every $0 \le i < n$ independently, gather all pairs $(i, W_s[i])$, with $0 \le s < n/k$, and return the pair $(i, W[i])$, with $W[i] = \sum_{s=0}^{n/k-1} W_s[i]$.

□

Exercise. Generalize the space-efficient matrix-vector multiplication and matrix multiplication algorithms presented in class to handle rectangular matrices.

Solution. Consider first the case of the matrix-vector multiplication $W = A \cdot V$, where A is an $n \times m$-matrix and V an m-vector. We assume that A is represented through the nm pairs $((i,j), A[i,j])$, with $0 \le i < n$ and $0 \le j < m$, where (i,j) is the key, and that V

(resp., W) is represented through the m (resp., n) pairs $(i, V[i])$, with $0 \leq i < m$ (resp., $(i, W[i])$, with $0 \leq i < n$), where i is the key. Let $k = \sqrt{m}$ and assume, for simplicity, that k is an integer. Consider V and each row $A^{(i)}$ of A subdivided into m/k segments of size k (namely, $V^{(s)}$ and $A^{(i,s)}$, with $0 \leq s < m/k$). For every $0 \leq i < n$ and $0 \leq s < m/k$, define $W_s[i] = A^{(i,s)} \cdot V^{(s)}$. Hence, entry $W[i]$ of W is given by

$$W[i] = \sum_{s=0}^{m/k-1} W_s[i].$$

A simple generalization of the algorithm presented in class for the case $m = n$ computes the product W in 2 rounds using local space $M_L = O(\sqrt{m})$ and aggregate space $M_A = O(nm)$. The details are left as an exercise.

Let us now look at the matrix multiplication $C = A \cdot B$, where A is an $n \times m$-matrix, B is an $m \times n$-matrix, and C is an $n \times n$-matrix. (The more general case is left as an exercise.) Consider A subdivided into $nm/(k_1 k_2)$ $k_1 \times k_2$ blocks, B subdivided into $mn/(k_1 k_2)$ $k_2 \times k_1$ blocks, and C subdivided into $n^2/(k_1)^2$ $k_1 \times k_1$ blocks, where each block of C is the sum of m/k_2 products of a block of A times a block of B. The values k_1 and k_2 are suitable design parameters. A simple generalization of the algorithm presented in class for the case $m = n$ computes the product C in 2 rounds. In the first round, all required products between the blocks of A and the blocks of B are computed in parallel. This implies, that each block of A or B must be replicated n/k_1 times. In the second round, for each entry of C the m/k_2, contributions are added up. Therefore, the algorithm requires local space sufficient to hold a block of A and a block of B, in the first round, and m/k_2 contributions to an entry of C, in the second round, that is $M_L = O(k_1 k_2 + m/k_2)$. As for the aggregate space, the replicas of the blocks of A and B yield $M_A = O(n^2 m/k_1)$. Setting k_1 and k_2 to suitable values one can exercise tradeoffs between local and aggregate space. The algorithm presented in class for the case of square matrices (i.e., $m = n$), implemented a special case of the above strategy with $k_1 = k_2 = n^{1/3}$. □