Esercises on MapReduce

Exercise. Specify input, output and intermediate key-value pairs for the space-efficient 2-round MR-algorithm for matrix-vector multiplication (see slides 18 and 19).[1] For simplicity, assume that k divides n.

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 represented through n^2 pairs $((i,j), A[i,j])$, with $0 \leq i,j < n$, where (i,j) is the key, and that V (resp., W) is represented through n pairs $(i, V[i])$ (resp., $(i, W[i])$), with $0 \leq i < n$, where i is the key.

Round 1

- **Map phase.** Each input pair $((i,j), A[i,j])$ is mapped into the intermediate pair $((s,t),((i,j), A[i,j]))$, with $s = \lfloor i/k \rfloor$ and $t = \lfloor j/k \rfloor$, where (s,t) is the key. Thus, submatrix $A^{(s,t)}$ will be represented by all such pairs whose key is (s,t). Also, input pair $(i, V[i])$ is mapped into the n/k intermediate pairs $((s,t),(i, V[i]))$ with $0 \leq s < n/k$ and $t = \lfloor i/k \rfloor$, where (s,t) is the key. Thus, the s-th replica of segment $V^{(t)}$ (denoted by $V_s^{(t)}$) will be represented by all such pairs whose key is (s,t).
- **Reduce phase.** For each $0 \leq s,t < n/k$ independently, the pairs represented $A^{(s,t)}$ and $V_s^{(t)}$ are gathered and the product $W_t^{(s)} = A^{(s,t)} \cdot V_s^{(t)}$ is computed. Recall that $W_t^{(s)}$ is the t-th contribution to the segment $W^{(s)}$, which comprises the entries $W[i]$ with $sk \leq i < (s+1)k$, and that $W^{(s)} = \sum_{t=0}^{n/k-1} W_t^{(s)}$. Therefore, we can simply represent $W_t^{(s)}$ through the s pairs $(i, W_t[i])$, with $sk \leq i < (s+1)k$.

Round 2

- **Map phase.** Identity map.
- **Reduce phase.** For every $0 \leq i < n$ independently, all pairs $(i, W_t[i])$, with $0 \leq t < n/k$, are gathered and the output pair $(i, W[i])$ is returned.

□

Exercise. Let T be a huge set of web pages gathered by a crawler. Design and analyze 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.

Solution. Suppose that T is provided in input as a set of key-value pairs (u_D, D) where D is a web page and u_D, the key of the pair, is the page URL. For each distinct word w occurring in some page, the algorithm must output a pair (w, L_w), whose key is the word w and whose value is the list L_w of URLs of all pages in T where w occurs. The task can be accomplished in one round, using the following Map and Reduce phases:

[1]MR-algorithm stands for MapReduce-algorithm.

- **Map phase.** Each pair (u_D, D) is mapped into a number of pairs (w, u_D), one for each distinct word $w \in D$.
- **Reduce phase.** For every word w (occurring in some page) independently, all pairs with key w, say $(w, u_1), (w, u_2), \ldots, (w, u_{\ell(w)})$, are gathered and the output pair (w, L_w) is computed, where L_w is the list of URLs $u_1, u_2, \ldots, u_{\ell(w)}$.

Let

$$\begin{aligned} N &= \text{total number of words (with repetitions) occurring in the pages} \\ m_1 &= \text{max size of a page (in number of words)} \\ m_2 &= \text{max number of pages containing a given word} \end{aligned}$$

It is immediate to see that the algorithm requires local space $M_L = \Theta(\max\{m_1, m_2\})$ and aggregate space $M_A = \Theta(N)$. □

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 $0 \le i < N$, is the key of the pair. While the trivial 1-round algorithm uses $\Theta(N)$ local space, the space requirements can be substantially lowered using the following simple 2-round strategy.

Round 1

- **Map phase.** Each pair (i, x_i) is mapped into the pair $(i \bmod \lfloor\sqrt{N}\rfloor, x_i)$.
- **Reduce phase.** For each key k independently, with $0 \le k < \lfloor\sqrt{N}\rfloor$, the set S_k of all intermediate pairs (k, x) is gathered and the pair $(0, \text{avg}_k))$ is produced, where $\text{avg}_k = (1/N)\sum_{(k,x)\in S_k} x$ is produced.

Round 2

- **Map phase.** Identity
- **Reduce phase.** All pairs $(0, \text{avg}_k)$ are gathered and the pair $(0, \text{avg})$, with $\text{avg} = \sum_{(0,\text{avg}_k)} \text{avg}_k$.

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

Observation. Suppose that the size of M_L is fixed. We can modify the above algorithm so that it uses local space 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 $\epsilon > 0$. The details are left as an exercise. $\square$

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 3-round algorithm. The 2-round algorithm does a simple removal of duplicates. It works as follows:

Round 1

- **Map phase.** Each pair (i, x_i) is mapped into the pair (x_i, i), that is, x_i becomes the key.
- **Reduce phase.** For each key x independently, the set S_x of all intermediate pairs (x, j) are gathered and a unique pair $(0, x)$ is produced.

Round 2

- **Map phase.** Identity
- **Reduce phase.** The set S_0 of all pairs $(0, x)$ is gathered and the pair $(0, \text{count})$, where $\text{count} = |S_0|$, is returned.

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 $K_1 \ge N/K_2$. It is immediate to see that the algorithm requires $M_L = \Theta\left(K_1 + K_2\right)$ and aggregate space $M_A = \Theta\left(N\right)$. Note that, since $K_1 \ge N/K_2$, we have that, in general, $M_L = \Omega\left(\sqrt{N}\right)$, but it may become much larger than $\sqrt{N}$ if K_2 is small.

Below, is an alternative strategy that reduces the local space requirements for small K_2, at the expense of an extra round.

Round 1

- **Map phase.** Each pair (i, x_i) is mapped into the pair $(i \bmod \lfloor\sqrt{N}\rfloor, x_i)$.
- **Reduce phase.** For each key k independently, with $0 \le k < \lfloor\sqrt{N}\rfloor$, the set S_k of all intermediate pairs (k, x) is gathered and the duplicates are eliminated.

Round 2

- **Map phase.** Each pair (k, x) is mapped into the pair (x, k), that is, x becomes the key.
- **Reduce phase.** For each x independently, the set S_x of all intermediate pairs (x, k) is gathered and only one pair $(0, x)$ is produced.

Round 3

- **Map phase.** Identity.
- **Reduce phase.** The set S_0 of all pairs $(0, x)$ is gathered and the pair $(0, \text{count})$, where $\text{count} = |S_0|$, is returned.

Let K_2 be defined as above. It is easy to see that Round 1 and 2 require local space $\Theta\left(\sqrt{N}\right)$, while Round 3 requires local space $\Theta(K_2)$. Therefore, the algorithm requires $M_L = \Theta\left(\sqrt{N} + K_2\right)$ and aggregate space $M_A = \Theta(N)$. For small K_2, namely, $K_2 \lll \sqrt{N}$, the algorithm uses substantially less local space than the previous one. □

Exercise. Consider a dataset T of N transactions over I given in input as in the SON algorithm. Assume that each transaction has constant legnth. 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?

Solution. Suppose that T is provided in input as a set of key-value pairs (i, t_i) where the key is a TID i, with $0 \le i < N$, and the value is a transaction t_i. We can extract the sample in one round by first choosing the indices of the transactions to be sampled in the map phase, and then extracting the transactions in the reduce phase. The details of the algorithm are as follows.

- **Map phase.** Each pair (i, t_i) is mapped into the pair $(i \bmod (N/K), (i, t_i))$, where the key is $i \bmod (N/K)$. Moreover, if $i < K$ the mapper will generate a random value, $x_i \in [0, N)$ and produce N/K pairs $(0, x_i), (1, x_i), \ldots, (N/K - 1, x_i)$.
- **Reduce phase.** For each key j independently, with $0 \le j < N/K$, gather the set T_j of all intermediate pairs $(j, (i, t_i))$ (i.e., with $j = i \bmod (N/K)$), and the set H_j of K, intermediate pairs (j, x_i) with $0 \le i < K$. Note that T_j and H_j contain K pairs each. Extract from T_j the multiset S_j of all pairs $(j, (i, t_i))$ such that i corresponds to some index x_ℓ with $\ell \in [0, K)$. Multiple copies of $(j, (i, t_i))$ will be included in S_j in case more than one of the indices $x_1, x_2, \ldots, x_\ell$ (which are chosen with replacement) are equal to i.

The final sample S will be the aggregation of all S_j's. It is immediate to see that the algorithm requires $M_L = \Theta(K)$ and aggregate space $M_A = \Theta(N)$. □

Exercise. Generalize the space-efficient matrix-vector algorithm to handle rectangular matrices

Solution. Suppose that we must compute the product $W = A \cdot V$, where A is an $n \times m$-matrix and V an m-vector. We assume that A is represented through nm pairs $((i,j), A[i,j])$, with $0 \leq i < n$ and $0 \leq j < m$, where (i,j) is the key, and that V (resp., W) is represented through 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 $k = \lfloor \sqrt{m} \rfloor$ and assume, for simplicity, that k divides m. Consider A subdivided into nm/k rectangular blocks $1 \times k$, which we denote as $A^{(i,t)}$ with $0 \leq i < n$ and $0 \leq t < m/k$. Also, consider V subdivided into m/k segments of lenght k, which we denote as $V^{(t)}$, with $0 \leq t < m/k$. In the first round, we compute independently the nm/k values $W_t[i] = A^{(i,t)} \cdot V^{(t)}$, with $0 \leq i < n$ and $0 \leq t < m/k$. In the second round, for every index i, with $0 \leq i < n$, we compute the entry $W[i]$ of the output vector as $W[i] = \sum_{t=0}^{m/k-1} W_t[i]$. It is easy to see that both rounds require local space $M_L = O\left(\sqrt{m}\right)$ and aggregate space $M_A = O\left(nm\right)$. The details are left as an exercise.

Observation. Note that for $m = n$ the above algorithm is more sapce-efficient than the one presented in class. $\square$

Exercise. In the SON algorithm the dataset T of N transactions is initially partitioned into K subsets T_j, with $0 \leq j < K$, so that the transaction of index i is assigned to subset T_j with $j = i \bmod K$. If the transactions have arbitrary lengths, this partition may be unbalanced in terms of space. Is there a better way to partition T?

Solution. Let $T = \{t_i \ : \ 0 \leq i < N\}$ and let ℓ_i be the length of t_i, for each $0 \leq i < N$. Therefore, the total space occupied by T is $M = \sum_{i=0}^{N-1} \ell_i$. We can assign each transaction $t_i \in T$ to T_j, where j is a random index in $[0, K)$, chosen with uniform probability. The expected *aggregate length* of the transactions assigned to T_j, for any arbitrary j, is $(1/K) \sum_{i=0}^{N-1} \ell_i = M/K$. $\square$