2.6 Complexity Theory for Map-Reduce

Now, we shall explore the design of map-reduce algorithms in more detail. Section 2.5 introduced the idea that communication between the Map and Reduce tasks often accounts for the largest fraction of the time spent by these tasks. Here, we shall look at how the communication cost relates to other desiderata for map-reduce algorithms, in particular our desire to shrink the wall-clock time and to execute each reducer in main memory. Recall that a "reducer" is the

execution of the Reduce function on a single key and its associated value list. The point of the exploration in this section is that for many problems there is a spectrum of map-reduce algorithms requiring different amounts of communication. Moreover, the less communication an algorithm uses, the worse it may be in other respects, including wall-clock time and the amount of main memory it requires.

2.6.1 Reducer Size and Replication Rate

Let us now introduce the two parameters that characterize families of map-reduce algorithms. The first is the *reducer size*, which we denote by q. This parameter is the upper bound on the number of values that are allowed to appear in the list associated with a single key. Reducer size can be selected with at least two goals in mind.

1. By making the reducer size small, we can force there to be many reducers, i.e., many different keys according to which the problem input is divided by the Map tasks. If we also create many Reduce tasks – even one for each reducer – then there will be a high degree of parallelism, and we can look forward to a low wall-clock time.

2. We can choose a reducer size sufficiently small that we are certain the computation associated with a single reducer can be executed entirely in the main memory of the compute node where its Reduce task is located. Regardless of the computation done by the reducers, the running time will be greatly reduced if we can avoid having to move data repeatedly between main memory and disk.

The second parameter is the *replication rate*, denoted r. We define r to be the number of key-value pairs produced by all the Map tasks on all the inputs, divided by the number of inputs. That is, the replication rate is the average communication from Map tasks to Reduce tasks (measured by counting key-value pairs) per input.

Example 2.11 : Let us consider the one-pass matrix-multiplication algorithm of Section 2.3.10. Suppose that all the matrices involved are $n \times n$ matrices. Then the replication rate r is equal to n. That fact is easy to see, since for each element m_{ij}, there are n key-value pairs produced; these have all keys of the form (i, k), for $1 \leq k \leq n$. Likewise, for each element of the other matrix, say n_{jk}, we produce n key-value pairs, each having one of the keys (i, k), for $1 \leq i \leq n$. In this case, not only is n the average number of key-value pairs produced for an input element, but each input produces exactly this number of pairs.

We also see that q, the required reducer size is $2n$. That is, for each key (i, k), there are n key-value pairs representing elements m_{ij} of the first matrix and another n key-value pairs derived from the elements n_{jk} of the second matrix.

While this pair of values represents only one particular algorithm for one-pass matrix multiplication, we shall see that it is part of a spectrum of algorithms, and in fact represents an extreme point, where q is as small as can be, and r is at its maximum. More generally, there is a tradeoff between r and q, that can be expressed as $qr \geq 2n^2$. □

2.6.2 An Example: Similarity Joins

To see the tradeoff between r and q in a realistic situation, we shall examine a problem known as *similarity join*. In this problem, we are given a large set of elements X and a similarity measure $s(x, y)$ that tells how similar two elements x and y of set X are. In Chapter 3 we shall learn about the most important notions of similarity and also learn some tricks that let us find similar pairs quickly. But here, we shall consider only the raw form of the problem, where we have to look at each pair of elements of X and determine their similarity by applying the function s. We assume that s is symmetric, so $s(x, y) = s(y, x)$, but we assume nothing else about s. The output of the algorithm is those pairs whose similarity exceeds a given threshold t.

For example, let us suppose we have a collection of one million images, each of size one megabyte. Thus, the dataset has size one terabyte. We shall not try to describe the similarity function s, but it might involve giving higher values when images have roughly the same distribution of colors and when images have corresponding regions with the same distribution of colors. The goal would be to discover pairs of images that show the same type of object or scene. This problem is extremely hard, but classifying by color distribution is generally of some help toward that goal.

Let us look at how we might do the computation using map-reduce to exploit the natural parallelism found in this problem. The input is key-value pairs (i, P_i), where i is an ID for the picture and P_i is the picture itself. We want to compare each pair of pictures, so let us use one key for each set of two ID's $\{i, j\}$. There are approximately 5×10^{11} pairs of two ID's. We want each key $\{i, j\}$ to be associated with the two values P_i and P_j, so the input to the corresponding reducer will be $(\{i, j\}, [R_i, R_j])$. Then, the Reduce function can simply apply the similarity function s to the two pictures on its value list, that is, compute $s(R_i, R_j)$, and decide whether the similarity of the two pictures is above threshold. The pair would be output if so.

Alas, this algorithm will fail completely. The reducer size is small, since no list has more than two values, or a total of 2MB of input. Although we don't know exactly how the similarity function s operates, we can reasonably expect that it will not require more than the available main memory. However, the replication rate is 999,999, since for each picture we generate that number of key-value pairs, one for each of the other pictures in the dataset. The total number of bytes communicated from Map tasks to Reduce tasks is 1,000,000 (for the pictures) times 999,999 (for the replication), times 1,000,000 (for the size of each picture). That's 10^{18} bytes, or one exabyte. To communicate this

amount of data over gigabit Ethernet would take 10^{10} seconds, or about 300 years.[9]

Fortunately, this algorithm is only the extreme point in a spectrum of possible algorithms. We can characterize these algorithms by grouping pictures into g groups, each of $10^6/g$ pictures.

The Map Function: Take an input element (i, R_i) and generate $g-1$ key-value pairs. For each, the key is one of the sets $\{u, v\}$, where u is the group to which picture i belongs, and v is one of the other groups. The associated value is the pair (i, R_i).

The Reduce Function: Consider the key $\{u, v\}$. The associated value list will have the $2 \times 10^6/g$ elements (j, R_j), where j belongs to either group u or group v. The Reduce function takes each (i, R_i) and (j, R_j) on this list, where i and j belong to different groups, and applies the similarity function $s(R_i, R_j)$. In addition, we need to compare the pictures that belong to the same group, but we don't want to do the same comparison at each of the $g-1$ reducers whose key contains a given group number. There are many ways to handle this problem, but one way is as follows. Compare the members of group u at the reducer $\{u, u+1\}$, where the "+1" is taken in the end-around sense. That is, if $u = g$ (i.e., u is the last group), then $u+1$ is group 1. Otherwise, $u+1$ is the group whose number is one greater than u.

We can compute the replication rate and reducer size as a function of the number of groups g. Each input element is turned into $g-1$ key-value pairs. That is, the replication rate is $g-1$, or approximately $r = g$, since we suppose that the number of groups is still fairly large. The reducer size is $2 \times 10^6/g$, since that is the number of values on the list for each reducer. Each value is about a megabyte, so the number of bytes needed to store the input is $2 \times 10^{12}/g$.

Example 2.12: If g is 1000, then the input consumes about 2GB. That's enough to hold everything in a typical main memory. Moreover, the total number of bytes communicated is now $10^6 \times 999 \times 10^6$, or about 10^{15} bytes. While that is still a huge amount of data to communicate, it is 1000 times less than that of the obvious algorithm. Moreover, there are still about half a million reducers. Since we are unlikely to have available that many compute nodes, we can divide all the reducers into a smaller number of Reduce tasks and still keep all the compute nodes busy; i.e., we can get as much parallelism as our computing cluster offers us. □

The computation cost for algorithms in this family is independent of the number of groups g, as long as the input to each reducer fits in main memory. The reason is that the bulk of the computation is the application of function s to the pairs of pictures. No matter what value g has, s is applied to each pair

[9]In a typical cluster, there are many switches connecting subsets of the compute nodes, so all the data does not need to go across a single gigabit switch. However, the total available communication is still small enough that it is not feasible to implement this algorithm for the scale of data we have hypothesized.

once and only once. Thus, although the work of algorithms in the family may be divided among reducers in widely different ways, all members of the family do the same computation.

2.6.3 A Graph Model for Map-Reduce Problems

In this section, we begin the study of a technique that will enable us to prove lower bounds on the replication rate, as a function of reducer size for a number of problems. Our first step is to introduce a graph model of problems. For each problem solvable by a map-reduce algorithm there is:

1. A set of inputs.
2. A set of outputs.
3. A many-many relationship between the inputs and outputs, which describes which inputs are necessary to produce which outputs.

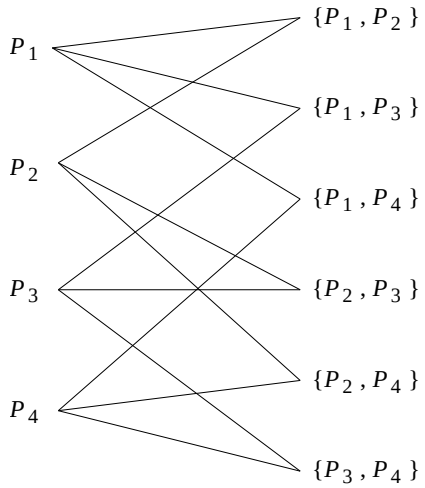

Figure 2.9: Input-output relationship for a similarity join

Example 2.13: Figure 2.9 shows the graph for the similarity-join problem discussed in Section 2.6.2, if there were four pictures rather than a million. The inputs are the pictures, and the outputs are the six possible pairs of pictures. Each output is related to the two inputs that are members of its pair. □

Example 2.14: Matrix multiplication presents a more complex graph. If we multiply $n \times n$ matrices M and N to get matrix P, then there are $2n^2$ inputs, m_{ij} and n_{jk}, and there are n^2 outputs p_{ik}. Each output p_{ik} is related to $2n$ inputs: $m_{i1}, m_{i2}, \ldots, m_{in}$ and $n_{1k}, n_{2k}, \ldots, n_{nk}$. Moreover, each input is related

to n outputs. For example, m_{ij} is related to $p_{i1}, p_{i2}, \ldots, p_{in}$. Figure 2.10 shows the input-output relationship for matrix multiplication for the simple case of 2×2 matrices, specifically

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

□

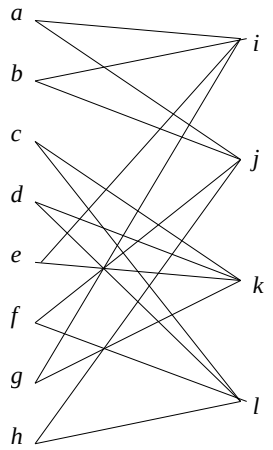

Figure 2.10: Input-output relationship for matrix multiplication

In the problems of Examples 2.13 and 2.14, the inputs and outputs were clearly all present. However, there are other problems where the inputs and/or outputs may not all be present in any instance of the problem. An example of such a problem is the natural join of $R(A, B)$ and $S(B, C)$ discussed in Section 2.3.7. We assume the attributes A, B, and C each have a finite domain, so there are only a finite number of possible inputs and outputs. The inputs are all possible R-tuples, those consisting of a value from the domain of A paired with a value from the domain of B, and all possible S-tuples – pairs from the domains of B and C. The outputs are all possible triples, with components from the domains of A, B, and C in that order. The output (a, b, c) is connected to two inputs, namely $R(a, b)$ and $S(b, c)$.

But in an instance of the join computation, only some of the possible inputs will be present, and therefore only some of the possible outputs will be produced. That fact does not influence the graph for the problem. We still need to know how every possible output relates to inputs, whether or not that output is produced in a given instance.

2.6.4 Mapping Schemas

Now that we see how to represent problems addressable by map-reduce as graphs, we can define the requirements for a map-reduce algorithm to solve

a given problem. Each such algorithm must have a *mapping schema*, which expresses how outputs are produced by the various reducers used by the algorithm. That is, a mapping schema for a given problem with a given reducer size q is an assignment of inputs to one or more reducers, such that:

1. No reducer is assigned more that q inputs.
2. For every output of the problem, there is at least one reducer that is assigned all the inputs that are related to that output. We say this reducer *covers* the output.

It can be argued that the existence of a mapping schema for any reducer size is what distinguishes problems that can be solved by a single map-reduce job from those that cannot.

Example 2.15: Let us reconsider the "grouping" strategy we discussed in connection with the similarity join in Section 2.6.2. To generalize the problem, suppose the input is p pictures, which we place in g equal-sized groups of p/g inputs each. The number of outputs is $\binom{p}{2}$, or approximately $p^2/2$ outputs. A reducer will get the inputs from two groups, that is $2p/g$ inputs, so the reducer size we need is $q = 2p/g$. Each picture is sent to the reducers corresponding to the pairs consisting of its group and any of the $g - 1$ other groups. Thus, the replication rate is $g - 1$, or approximately g. If we replace g by the replication rate r in $q = 2p/g$, we conclude that $r = 2p/q$. That is, the replication rate is inversely proportional to the reducer size. That relationship is common; the smaller the reducer size, the larger the replication rate, and therefore the higher the communication.

This family of algorithms is described by a family of mapping schemas, one for each possible q. In the mapping schema for $q = 2p/g$, there are $\binom{g}{2}$, or approximately $g^2/2$ reducers. Each reducer corresponds to a pair of groups, and an input P is assigned to all the reducers whose pair includes the group of P. Thus, no reducer is assigned more than $2p/g$ inputs; in fact each reducer is assigned exactly that number. Moreover, every output is covered by some reducer. Specifically, if the output is a pair from two different groups u and v, then this output is covered by the reducer for the pair of groups $\{u, v\}$. If the output corresponds to inputs from only one group u, then the output is covered by several reducers – those corresponding to the set of groups $\{u, v\}$ for any $v \neq u$. Note that the algorithm we described has only one of these reducers computing the output, but any of them *could* compute it. □

The fact that an output depends on a certain input means that when that input is processed at the Map task, there will be at least one key-value pair generated to be used when computing that output. The value might not be exactly the input (as was the case in Example 2.15), but it is derived from that input. What is important is that for every related input and output there is a unique key-value pair that must be communicated. Note that there is

technically never a need for more than one key-value pair for a given input and output, because the input could be transmitted to the reducer as itself, and whatever transformations on the input were applied by the Map function could instead be applied by the Reduce function at the reducer for that output.

2.6.5 When Not All Inputs Are Present

Example 2.15 describes a problem where we know every possible input is present, because we can define the input set to be those pictures that actually exist in the dataset. However, as discussed at the end of Section 2.6.3, there are problems like computing the join, where the graph of inputs and outputs describes inputs that might exist, and outputs that are only made when at least one of the inputs exists in the dataset. In fact, for the join, both inputs related to an output must exist if we are to make that output.

An algorithm for a problem where outputs can be missing still needs a mapping schema. The justification is that all inputs, or any subset of them, *might* be present, so an algorithm without a mapping schema would not be able to produce every possible output if all the inputs related to that output happened to be present, and yet no reducer covered that output.

The only way the absence of some inputs makes a difference is that we may wish to rethink the desired value of the reducer size q when we select an algorithm from the family of possible algorithms. Especially, if the value of q we select is that number such that we can be sure the input will just fit in main memory, then we may wish to increase q to take into account that some fraction of the inputs are not really there.

Example 2.16: Suppose that we know we can execute the Reduce function in main memory on a key and its associated list of q values. However, we also know that only 5% of the possible inputs are really present in the data set. Then a mapping schema for reducer size q will really send about $q/20$ of the inputs that exist to each reducer. Put another way, we could use the algorithm for reducer size $20q$ and expect that an average of q inputs will actually appear on the list for each reducer. We can thus choose $20q$ as the reducer size, or since there will be some randomness in the number of inputs actually appearing at each reducer, we might wish to pick a slightly smaller value of reducer size, such as $18q$. □

2.6.6 Lower Bounds on Replication Rate

The family of similarity-join algorithms described in Example 2.15 lets us trade off communication against the reducer size, and through reducer size to trade communication against parallelism or against the ability to execute the Reduce function in main memory. How do we know we are getting the best possible tradeoff? We can only know we have the minimum possible communication if we can prove a matching lower bound. Using existence of a mapping schema as

the starting point, we can often prove such a lower bound. Here is an outline of the technique.

1. Prove an upper bound on how many outputs a reducer with q inputs can cover. Call this bound $g(q)$. This step can be difficult, but for examples like similarity join, it is actually quite simple.
2. Determine the total number of outputs produced by the problem.
3. Suppose that there are k reducers, and the ith reducer has $q_i < q$ inputs. Observe that $\sum_{i=1}^{k} g(q_i)$ must be no less than the number of outputs computed in step (2).
4. Manipulate the inequality from (3) to get a lower bound on $\sum_{i=1}^{k} q_i$. Often, the trick used at this step is to replace some factors of q_i by their upper bound q, but leave a single factor of q_i in the term for i.
5. Since $\sum_{i=1}^{k} q_i$ is the total communication from Map tasks to Reduce tasks, divide the upper bound from (4) on this quantity by the number of inputs. The result is the replication rate.

Example 2.17: This sequence of steps may seem mysterious, but let us consider the similarity join as an example that we hope will make things clear. Recall that in Example 2.15 we gave an upper bound on the replication rate r of $r \leq 2p/q$, where p was the number of inputs and q was the reducer size. We shall show a lower bound on r that is half that amount, which implies that, although improvements to the algorithm might be possible, any reduction in communication for a given reducer size will be by a factor of 2 at most.

For step (1), observe that if a reducer gets q inputs, it cannot cover more than $\binom{q}{2}$, or approximately $q^2/2$ outputs. For step (2), we know there are a total of $\binom{p}{2}$, or approximately $p^2/2$ outputs that each must be covered. The inequality constructed at step (3) is thus

$$\sum_{i=1}^{k} q_i^2/2 \leq p^2/2$$

or, multiplying both sides by 2,

$$\sum_{i=1}^{k} q_i^2 \geq p^2 \tag{2.1}$$

Now, we must do the manipulation of step (4). Following the hint, we note that there are two factors of q_i in each term on the left of Equation (2.1), so we replace one factor by q and leave the other as q_i. Since $q \geq q_i$, we can only increase the left side by doing so, and thus the inequality continues to hold:

$$q \sum_{i=1}^{k} q_i \geq p^2$$

or, dividing by q:

$$\sum_{i=1}^{k} q_i \geq p^2/q \tag{2.2}$$

The last step, which is step (5), is to divide both sides of Equation 2.2 by p, the number of inputs. As a result, the left side, which is $(\sum_{i=1}^{k} q_i)/p$ is equal to the replication rate, and the right side becomes p/q. That is, we have proved the lower bound on r:

$$r \geq p/q$$

As claimed, this shows that the family of algorithms from Example 2.15 all have a replication rate that is at most twice the lowest possible replication rate. □

2.6.7 Case Study: Matrix Multiplication

In this section we shall apply the lower-bound technique to one-pass matrix-multiplication algorithms. We saw one such algorithm in Section 2.3.10, but that is only an extreme case of a family of possible algorithms. In particular, for that algorithm, a reducer corresponds to a single element of the output matrix. Just as we grouped inputs in the similarity-join problem to reduce the communication at the expense of a larger reducer size, we can group rows and columns of the two input matrices into bands. Each pair consisting of a band of rows of the first matrix and a band of columns of the second matrix is used by one reducer to produce a square of elements of the output matrix. An example is suggested by Fig. 2.11.

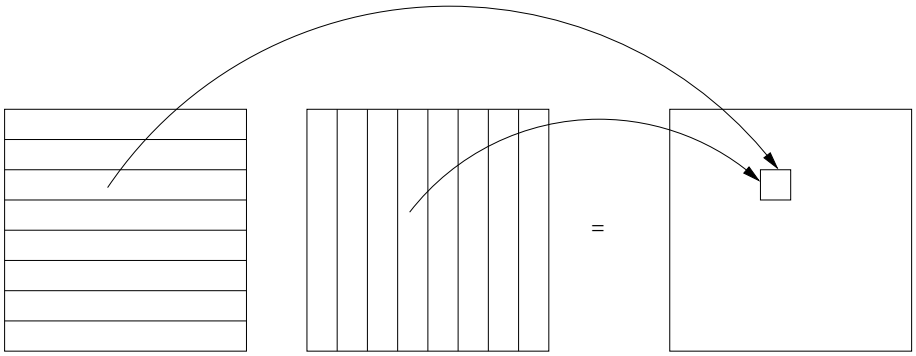

Figure 2.11: Dividing matrices into bands to reduce communication

In more detail, suppose we want to compute $MN = P$, and all three matrices are $n \times n$. Group the rows of M into g bands of n/g rows each, and group the columns of N into g bands of n/g columns each. This grouping is as suggested by Fig. 2.11. Keys correspond to two groups (bands), one from M and one from N.

The Map Function: For each element of M, the Map function generates g key-value pairs. The value in each case is the element itself, together with its row and column number so it can be identified by the Reduce function. The key is the group to which the element belongs, paired with any of the groups of the matrix N. Similarly, for each element of N, the Map function generates g key-value pairs. The key is the group of that element paired with any of the groups of M, and the value is the element itself plus its row and column.

The Reduce Function: The reducer corresponding to the key (i, j), where i is a group of M and j is a group of N, gets a value list consisting of all the elements in the ith band of M and the jth band of N. It thus has all the values it needs to compute the elements of P whose row is one of those rows comprising the ith band of M and whose column is one of those comprising the jth band of N. For instance, Fig. 2.11 suggests the third group of M and the fourth group of N, combining to compute a square of P at the reducer $(3, 4)$.

Each reducer gets $n(n/g)$ elements from each of the two matrices, so $q = 2n^2/g$. The replication rate is g, since each element of each matrix is sent to g reducers. That is, $r = g$. Combining $r = g$ with $q = n^2/g$ we can conclude that $r = n^2/q$. That is, just as for similarity join, the replication rate varies inversely with the reducer size.

It turns out that this upper bound on replication rate is also a lower bound. That is, we cannot do better than the family of algorithms we described above in a single round of map-reduce. Interestingly, we shall see that we can get a lower total communication for the same reducer size, if we use two passes of map-reduce as we discussed in Section 2.3.9. We shall not give the complete proof of the lower bound, but will suggest the important elements.

For step (1) we need to get an upper bound on how many outputs a reducer of size q can cover. First, notice that if a reducer gets some of the elements in a row of M, but not all of them, then the elements of that row are useless; the reducer cannot produce any output in that row of P. Similarly, if a reducer receives some but not all of a column of N, these inputs are also useless. Thus, we may assume that the best mapping schema will send to each reducer some number of full rows of M and some number of full columns of N. This reducer is then capable of producing output element p_{ik} if and only if it has received the entire ith row of M and the entire kth column of N. The remainder of the argument for step (1) is to prove that the largest number of outputs are covered when the reducer receives the same number of rows as columns. We leave this part as an exercise.

However, assuming a reducer receives k rows of M and k columns of N, then $q = 2nk$, and k^2 outputs are covered. That is, $g(q)$, the maximum number of outputs covered by a reducer that receives q inputs, is $q^2/4n^2$.

For step (2), we know the number of outputs is n^2. In step (3) we observe

that if there are k reducers, with the ith reducer receiving $q_i \leq q$ inputs, then

$$\sum_{i=1}^{k} q_i^2/4n^2 \geq n^2$$

or

$$\sum_{i=1}^{k} q_i^2 \geq 4n^4$$

From this inequality, you can derive

$$r \geq 2n^2/q$$

We leave the algebraic manipulation, which is similar to that in Example 2.17, as an exercise.

Now, let us consider the generalization of the two-pass matrix-multiplication algorithm that we described in Section 2.3.9. This algorithm too can be generalized so that the first map-reduce job uses reducers of size greater than 2. The idea is suggested by Fig. 2.12. We may divide the rows and columns of both input matrices M and N into g groups of n/g rows or columns each. The intersections of the groups partition each matrix into g^2 squares of n^2/g^2 elements each.

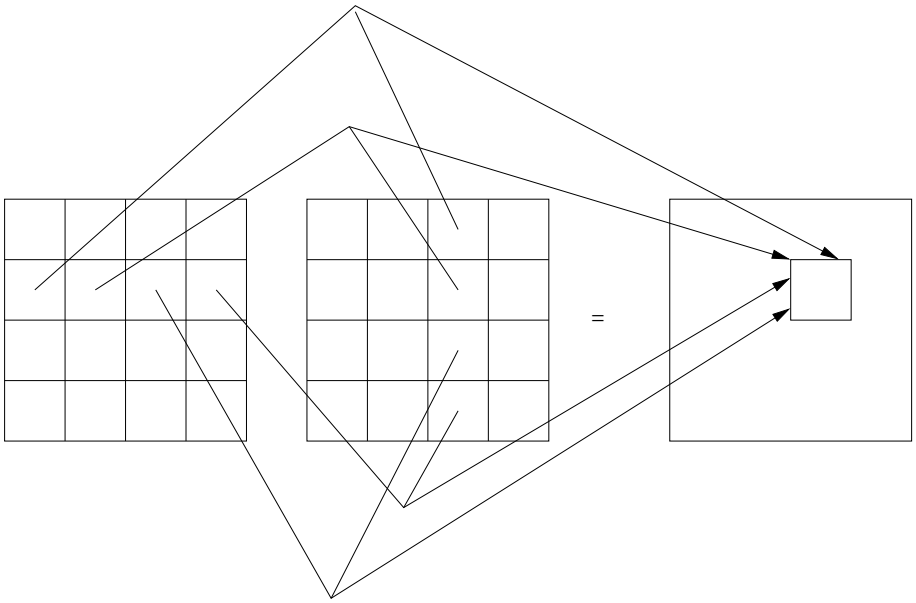

Figure 2.12: Partitioning matrices into squares for a two-pass map-reduce algorithm

The square of M corresponding to set of rows I and set of columns J combines with the square of N corresponding to set of rows J and set of columns K. These two squares compute some of the terms that are needed to produce the square of the output matrix P that has set of rows I and set of columns K. However, these two squares do not compute the full value of these elements of P; rather they produce a contribution to the sum. Other pairs of squares, one from M and one from N, contribute to the same square of P. These contributions are suggested in Fig. 2.12. There, we see how all the squares of M with a fixed value for set of rows I pair with all the squares of N that have a fixed value for the set of columns K by letting the set J vary.

So in the first pass, we compute the products of the square (I, J) of M with the square (J, K) of N, for all I, J, and K. Then, in the second pass, for each I and K we sum the products over all possible sets J. In more detail, the first map-reduce job does the following.

The Map Function: The keys are triples of sets of rows and/or column numbers (I, J, K). Suppose the element m_{ij} belongs to group of rows I and group of columns J. Then from m_{ij} we generate g key-value pairs with value equal to m_{ij}, together with its row and column numbers, i and j, to identify the matrix element. There is one key-value pair for each key (I, J, K), where K can be any of the g groups of columns of N. Similarly, from element n_{jk} of N, if j belongs to group J and k to group K, the Map function generates g key-value pairs with value consisting of n_{jk}, j, and k, and with keys (I, J, K) for any group I.

The Reduce Function: The reducer corresponding to (I, J, K) receives as input all the elements m_{ij} where i is in I and j is in J, and it also receives all the elements n_{jk}, where j is in J and k is in K. It computes

$$x_{iJk} = \sum_{j \text{ in } J} m_{ij} n_{jk}$$

for all i in I and k in K.

Notice that the replication rate for the first map-reduce job is g, and the total communication is therefore $2gn^2$. Also notice that each reducer gets $2n^2/g^2$ inputs, so $q = 2n^2/g^2$. Equivalently, $g = n\sqrt{2/q}$. Thus, the total communication $2gn^2$ can be written in terms of q as $2\sqrt{2}n^3/\sqrt{q}$.

The second map-reduce job is simple; it sums up the x_{iJk}'s over all sets J.

The Map Function: We assume that the Map tasks execute at whatever compute nodes executed the Reduce tasks of the previous job. Thus, no communication is needed between the jobs. The Map function takes as input one element x_{iJk}, which we assume the previous reducers have left labeled with i and k so we know to what element of matrix P this term contributes. One key-value pair is generated. The key is (i, k) and the value is x_{iJk}.

The Reduce Function: The reduce function simply sums the values associated with key (i, k) to compute the output element P_{ik}.

The communication between the Map and Reduce tasks of the second job is gn^2, since there are n possible values of i, n possible values of k, and g possible values of the set J, and each x_{iJk} is communicated only once. If we recall from our analysis of the first map-reduce job that $g = n\sqrt{2/q}$, we can write the communication for the second job as $n^2g = \sqrt{2}n^3/\sqrt{q}$. This amount is exactly half the communication for the first job, so the total communication for the two-pass algorithm is $3\sqrt{2}n^2/\sqrt{q}$. Although we shall not examine this point here, it turns out that we can do slightly better if we divide the matrices M and N not into squares but into rectangles that are twice as long on one side as on the other. In that case, we get the slightly smaller constant 4 in place of $3\sqrt{2} = 4.24$, and we get a two-pass algorithm with communication equal to $4n^3/\sqrt{q}$.

Now, recall that the replication rate we computed for the one-pass algorithm is $4n^4/q$. We may as well assume q is less than n^2, or else we can just use a serial algorithm at one compute node and not use map-reduce at all. Thus, $n^3/\sqrt{q}$ is smaller than n^4/q, and if q is close to its minimum possible value of $2n$,[10] then the two-pass algorithm beats the one-pass algorithm by a factor of $O(\sqrt{n})$ in communication. Moreover, we can expect the difference in communication to be the significant cost difference. Both algorithms do the same $O(n^3)$ arithmetic operations. The two-pass method naturally has more overhead managing tasks than does the one-job method. On the other hand, the second pass of the two-pass algorithm applies a Reduce function that is associative and commutative. Thus, it might be possible to save some communication cost by using a combiner on that pass.

2.6.8 Exercises for Section 2.6

Exercise 2.6.1: Describe the graphs that model the following problems.

(a) The multiplication of an $n \times n$ matrix by a vector of length n.

(b) The natural join of $R(A, B)$ and $S(B, C)$, where A, B, and C have domains of sizes a, b, and c, respectively.

(c) The grouping and aggregation on the relation $R(A, B)$, where A is the grouping attribute and B is aggregated by the MAX operation. Assume A and B have domains of size a and b, respectively.

! **Exercise 2.6.2:** Provide the details of the proof that a one-pass matrix-multiplication algorithm requires replication rate at least $r \geq 2n^2/q$, including:

(a) The proof that, for a fixed reducer size, the maximum number of outputs are covered by a reducer when that reducer receives an equal number of rows of M and columns of N.

[10] If q is less than $2n$, then a reducer cannot get even one row and one column, and therefore cannot compute any outputs at all.

(b) The algebraic manipulation needed, starting with $\sum_{i=1}^{k} q_i^2 \geq 4n^4$.

!! **Exercise 2.6.3:** Suppose our inputs are bit strings of length b, and the outputs correspond to pairs of strings at Hamming distance 1.[11]

(a) Prove that a reducer of size q can cover at most $(q/2)\log_2 q$ outputs.

(b) Use part (a) to show the lower bound on replication rate: $r \geq b/\log_2 q$.

(c) Show that there are algorithms with replication rate as given by part (b) for the cases $q = 2$, $q = 2^b$, and $q = 2^{b/2}$.

[11] Bit strings have *Hamming distance 1* if they differ in exactly one bit position. You may look ahead to Section 3.5.6 for the general definition.