2.5 The Communication Cost Model

In this section we shall introduce a model for measuring the quality of algorithms implemented on a computing cluster of the type so far discussed in this chapter. We assume the computation is described by an acyclic workflow, as discussed in Section 2.4.1. For many applications, the bottleneck is moving data among tasks, such as transporting the outputs of Map tasks to their proper Reduce tasks. As an example, we explore the computation of multiway joins as single map-reduce jobs, and we see that in some situations, this approach is more efficient than the straightforward cascade of 2-way joins.

2.5.1 Communication-Cost for Task Networks

Imagine that an algorithm is implemented by an acyclic network of tasks. These tasks could be Map tasks feeding Reduce tasks, as in a standard map-reduce algorithm, or they could be several map-reduce jobs cascaded, or a more general workflow structure, such as a collection of tasks each of which implements the workflow of Fig. 2.6.[7] The *communication cost* of a task is the size of the input to the task. This size can be measured in bytes. However, since we shall be using relational database operations as examples, we shall often use the number of tuples as a measure of size.

[7]Recall that this figure represented functions, not tasks. As a network of tasks, there would be, for example, many tasks implementing function f, each of which feeds data to each of the tasks for function g and each of the tasks for function i.

The *communication cost of an algorithm* is the sum of the communication cost of all the tasks implementing that algorithm. We shall focus on the communication cost as the way to measure the efficiency of an algorithm. In particular, we do not consider the amount of time it takes each task to execute when estimating the running time of an algorithm. While there are exceptions, where execution time of tasks dominates, these exceptions are rare in practice. We can explain and justify the importance of communication cost as follows.

- The algorithm executed by each task tends to be very simple, often linear in the size of its input.
- The typical interconnect speed for a computing cluster is one gigabit per second. That may seem like a lot, but it is slow compared with the speed at which a processor executes instructions. Moreover, in many cluster architectures, there is competition for the interconnect when several compute nodes need to communicate at the same time. As a result, the compute node can do a lot of work on a received input element in the time it takes to deliver that element.
- Even if a task executes at a compute node that has a copy of the chunk(s) on which the task operates, that chunk normally will be stored on disk, and the time taken to move the data into main memory may exceed the time needed to operate on the data once it is available in memory.

Assuming that communication cost is the dominant cost, we might still ask why we count only input size, and not output size. The answer to this question involves two points:

1. If the output of one task τ is input to another task, then the size of τ's output will be accounted for when measuring the input size for the receiving task. Thus, there is no reason to count the size of any output except for those tasks whose output forms the result of the entire algorithm.
2. But in practice, the algorithm output is rarely large compared with the input or the intermediate data produced by the algorithm. The reason is that massive outputs cannot be used unless they are summarized or aggregated in some way. For example, although we talked in Example 2.6 of computing the entire transitive closure of a graph, in practice we would want something much simpler, such as the count of the number of nodes reachable from each node, or the set of nodes reachable from a single node.

Example 2.8: Let us evaluate the communication cost for the join algorithm from Section 2.3.7. Suppose we are joining $R(A, B) \bowtie S(B, C)$, and the sizes of relations R and S are r and s, respectively. Each chunk of the files holding R and S is fed to one Map task, so the sum of the communication costs for all the Map tasks is $r + s$. Note that in a typical execution, the Map tasks will

each be executed at a compute node holding a copy of the chunk to which it applies. Thus, no internode communication is needed for the Map tasks, but they still must read their data from disk. Since all the Map tasks do is make a simple transformation of each input tuple into a key-value pair, we expect that the computation cost will be small compared with the communication cost, regardless of whether the input is local to the task or must be transported to its compute node.

The sum of the outputs of the Map tasks is roughly as large as their inputs. Each output key-value pair is sent to exactly one Reduce task, and it is unlikely that this Reduce task will execute at the same compute node. Therefore, communication from Map tasks to Reduce tasks is likely to be across the interconnect of the cluster, rather than memory-to-disk. This communication is $O(r + s)$, so the communication cost of the join algorithm is $O(r + s)$.

The Reduce tasks execute the reducer (application of the Reduce function to a key and its associated value list) for one or more values of attribute B. Each reducer takes the inputs it receives and divides them between tuples that came from R and those that came from S. Each tuple from R pairs with each tuple from S to produce one output. The output size for the join can be either larger or smaller than $r + s$, depending on how likely it is that a given R-tuple joins with a given S-tuple. For example, if there are many different B-values, we would expect the output to be small, while if there are few B-values, a large output is likely.

If the output is large, then the computation cost of generating all the outputs from a reducer could be much larger than $O(r+s)$. However, we shall rely on our supposition that if the output of the join is large, then there is probably some aggregation being done to reduce the size of the output. It will be necessary to communicate the result of the join to another collection of tasks that perform this aggregation, and thus the communication cost will be at least proportional to the computation needed to produce the output of the join. □

2.5.2 Wall-Clock Time

While communication cost often influences our choice of algorithm to use in a cluster-computing environment, we must also be aware of the importance of *wall-clock time*, the time it takes a parallel algorithm to finish. Using careless reasoning, one could minimize total communication cost by assigning all the work to one task, and thereby minimize total communication. However, the wall-clock time of such an algorithm would be quite high. The algorithms we suggest, or have suggested so far, have the property that the work is divided fairly among the tasks. Therefore, the wall-clock time would be approximately as small as it could be, given the number of compute nodes available.

2.5.3 Multiway Joins

To see how analyzing the communication cost can help us choose an algorithm in the cluster-computing environment, we shall examine carefully the case of a multiway join. There is a general theory in which we:

1. Select certain attributes of the relations involved in the natural join of three or more relations to have their values hashed, each to some number of buckets.

2. Select the number of buckets for each of these attributes, subject to the constraint that the product of the numbers of buckets for each attribute is k, the number of reducers that will be used.

3. Identify each of the k reducers with a vector of bucket numbers. These vectors have one component for each of the attributes selected at step (1).

4. Send tuples of each relation to all those reducers where it might find tuples to join with. That is, the given tuple t will have values for some of the attributes selected at step (1), so we can apply the hash function(s) to those values to determine certain components of the vector that identifies the reducers. Other components of the vector are unknown, so t must be sent to reducers for all vectors having any value in these unknown components.

Some examples of this general technique appear in the exercises.

Here, we shall look only at the join $R(A,B) \bowtie S(B,C) \bowtie T(C,D)$ as an example. Suppose that the relations R, S, and T have sizes r, s, and t, respectively, and for simplicity, suppose that the probability is p that

1. An R-tuple and and S-tuple agree on B, and also the probability that

2. An S-tuple and a T-tuple agree on C.

If we join R and S first, using the map-reduce algorithm of Section 2.3.7, then the communication cost is $O(r+s)$, and the size of the intermediate join $R \bowtie S$ is prs. When we join this result with T, the communication of this second map-reduce job is $O(t+prs)$. Thus, the entire communication cost of the algorithm consisting of two 2-way joins is $O(r+s+t+prs)$. If we instead join S and T first, and then join R with the result, we get another algorithm whose communication cost is $O(r+s+t+pst)$.

A third way to take this join is to use a single map-reduce job that joins the three relations at once. Suppose that we plan to use k reducers for this job. Pick numbers b and c representing the number of buckets into which we shall hash B- and C-values, respectively. Let h be a hash function that sends B-values into b buckets, and let g be another hash function that sends C-values into c buckets. We require that $bc = k$; that is, each reducer corresponds to a pair of buckets, one for the B-value and one for the C-value. The reducer

corresponding to bucket pair (i, j) is responsible for joining the tuples $R(u, v)$, $S(v, w)$, and $T(w, x)$ whenever $h(v) = i$ and $g(w) = j$.

As a result, the Map tasks that send tuples of R, S, and T to the reducers that need them must send R- and T-tuples to more than one reducer. For an S-tuple $S(v, w)$, we know the B- and C-values, so we can send this tuple only to the reducer for $\big(h(v), g(w)\big)$. However, consider an R-tuple $R(u, v)$. We know it only needs to go to reducers that correspond to $\big(h(v), y\big)$, for some y. But we don't know y; the value of C could be anything as far as we know. Thus, we must send $R(u, v)$ to c reducers, since y could be any of the c buckets for C-values. Similarly, we must send the T-tuple $T(w, x)$ to each of the reducers $\big(z, g(w)\big)$ for any z. There are b such reducers.

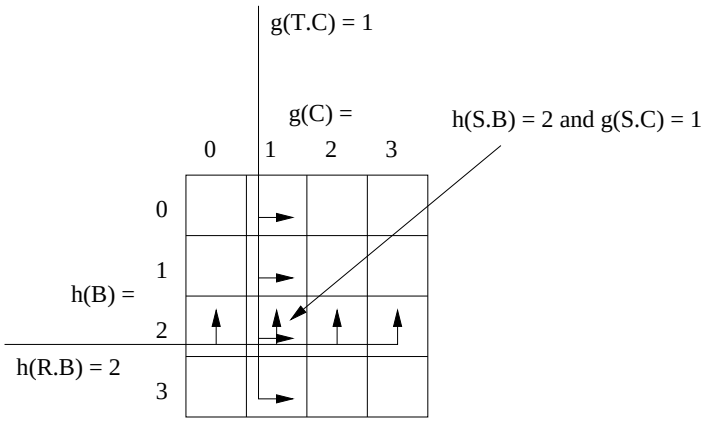

Figure 2.8: Sixteen reducers together perform a 3-way join

Example 2.9: Suppose that $b = c = 4$, so $k = 16$. The sixteen reducers can be thought of as arranged in a rectangle, as suggested by Fig. 2.8. There, we see a hypothetical S-tuple $S(v, w)$ for which $h(v) = 2$ and $g(w) = 1$. This tuple is sent by its Map task only to the reducer for key $(2, 1)$. We also see an R-tuple $R(u, v)$. Since $h(v) = 2$, this tuple is sent to all reducers $(2, y)$, for $y = 1, 2, 3, 4$. Finally, we see a T-tuple $T(w, x)$. Since $g(w) = 1$, this tuple is sent to all reducers $(z, 1)$ for $z = 1, 2, 3, 4$. Notice that these three tuples join, and they meet at exactly one reducer, the reducer for key $(2, 1)$. □

Now, suppose that the sizes of R, S, and T are different; recall we use r, s, and t, respectively, for those sizes. If we hash B-values to b buckets and C-values to c buckets, where $bc = k$, then the total communication cost for moving the tuples to the proper reducers is the sum of:

1. s to move each tuple $S(v, w)$ once to the reducer $\big(h(v), g(w)\big)$.
2. cr to move each tuple $R(u, v)$ to the c reducers $\big(h(v), y\big)$ for each of the c possible values of y.

Computation Cost of the 3-Way Join

Each of the reducers must join of parts of the three relations, and it is reasonable to ask whether this join can be taken in time that is linear in the size of the input to that Reduce task. While more complex joins might not be computable in linear time, the join of our running example can be executed at each Reduce process efficiently. First, create an index on $R.B$, to organize the R-tuples received. Likewise, create an index on $T.C$ for the T-tuples. Then, consider each received S-tuple, $S(v, w)$. Use the index on $R.B$ to find all R-tuples with $R.B = v$ and use the index on $T.C$ to find all T-tuples with $T.C = w$.

3. bt to move each tuple $T(w, x)$ to the b reducers $\big(z, g(w)\big)$ for each of the b possible values of z.

There is also a cost $r + s + t$ to make each tuple of each relation be input to one of the Map tasks. This cost is fixed, independent of b, c, and k.

We must select b and c, subject to the constraint $bc = k$, to minimize $s + cr + bt$. We shall use the technique of Lagrangean multipliers to find the place where the function $s + cr + bt - \lambda(bc - k)$ has its derivatives with respect to b and c equal to 0. That is, we must solve the equations $r - \lambda b = 0$ and $t - \lambda c = 0$. Since $r = \lambda b$ and $t = \lambda c$, we may multiply corresponding sides of these equations to get $rt = \lambda^2 bc$. Since $bc = k$, we get $rt = \lambda^2 k$, or $\lambda = \sqrt{rt/k}$. Thus, the minimum communication cost is obtained when $c = t/\lambda = \sqrt{kt/r}$, and $b = r/\lambda = \sqrt{kr/t}$.

If we substitute these values into the formula $s + cr + bt$, we get $s + 2\sqrt{krt}$. That is the communication cost for the Reduce tasks, to which we must add the cost $s + r + t$ for the communication cost of the Map tasks. The total communication cost is thus $r + 2s + t + 2\sqrt{krt}$. In most circumstances, we can neglect $r + t$, because it will be less than $2\sqrt{krt}$, usually by a factor of $O(\sqrt{k})$.

Example 2.10: Let us see under what circumstances the 3-way join has lower communication cost than the cascade of two 2-way joins. To make matters simple, let us assume that R, S, and T are all the same relation R, which represents the "friends" relation in a social network like Facebook. There are roughly a billion subscribers on Facebook, with an average of 300 friends each, so relation R has $r = 3 \times 10^{11}$ tuples. Suppose we want to compute $R \bowtie R \bowtie R$, perhaps as part of a calculation to find the number of friends of friends of friends each subscriber has, or perhaps just the person with the largest number of friends of friends of friends.[8] The cost of the 3-way join of R with itself is $4r + 2r\sqrt{k}$; $3r$ represents the cost of the Map tasks, and $r + 2\sqrt{kr^2}$ is the cost

[8]This person, or more generally, people with large extended circles of friends, are good people to use to start a marketing campaign by giving them free samples.

of the Reduce tasks. Since we assume $r = 3 \times 10^{11}$, this cost is $1,2 \times 10^{12} + 6 \times 10^{11}\sqrt{k}$.

Now consider the communication cost of joining R with itself, and then joining the result with R again. The Map and Reduce tasks for the first join each have a cost of $2r$, so the first join only has communication cost $4r = 1.2 \times 10^{12}$. But the size of $R \bowtie R$ is large. We cannot say exactly how large, since friends tend to fall into cliques, and therefore a person with 300 friends will have many fewer than the maximum possible number of friends of friends, which is 90,000. Let us estimate conservatively that the size of $R \bowtie R$ is not $300r$, but only $30r$, or 9×10^{12}. The communication cost for the second join of $(R \bowtie R) \bowtie R$ is thus $1.8 \times 10^{13} + 6 \times 10^{11}$. The total cost of the two joins is therefore $1.2 \times 10^{12} + 1.8 \times 10^{13} + 6 \times 10^{11} = 1.98 \times 10^{13}$.

We must ask whether the cost of the 3-way join, which is

$$1.2 \times 10^{12} + 6 \times 10^{11}\sqrt{k}$$

is less than 1.98×10^{13}. That is so, provided $6 \times 10^{11}\sqrt{k} < 1.86 \times 10^{13}$, or $\sqrt{k} < 31$. That is, the 3-way join will be preferable provided we use no more than $31^2 = 961$ reducers. □

2.5.4 Exercises for Section 2.5

Exercise 2.5.1: What is the communication cost of each of the following algorithms, as a function of the size of the relations, matrices, or vectors to which they are applied?

(a) The matrix-vector multiplication algorithm of Section 2.3.2.

(b) The union algorithm of Section 2.3.6.

(c) The aggregation algorithm of Section 2.3.8.

(d) The matrix-multiplication algorithm of Section 2.3.10.

! **Exercise 2.5.2:** Suppose relations R, S, and T have sizes r, s, and t, respectively, and we want to take the 3-way join $R(A, B) \bowtie S(B, C) \bowtie T(A, C)$, using k reducers. We shall hash values of attributes A, B, and C to a, b, and c buckets, respectively, where $abc = k$. Each reducer is associated with a vector of buckets, one for each of the three hash functions. Find, as a function of r, s, t, and k, the values of a, b, and c that minimize the communication cost of the algorithm.

! **Exercise 2.5.3:** Suppose we take a star join of a fact table $F(A_1, A_2, \ldots, A_m)$ with dimension tables $D_i(A_i, B_i)$ for $i = 1, 2, \ldots, m$. Let there be k reducers, each associated with a vector of buckets, one for each of the key attributes $A_1, A_2, \ldots, A_m$. Suppose the number of buckets into which we hash A_i is a_i.

Star Joins

A common structure for data mining of commercial data is the *star join*. For example, a chain store like Walmart keeps a *fact table* whose tuples each represent a single sale. This relation looks like $F(A_1, A_2, \ldots)$, where each attribute A_i is a key representing one of the important components of the sale, such as the purchaser, the item purchased, the store branch, or the date. For each key attribute there is a *dimension table* giving information about the participant. For instance, the dimension table $D(A_1, B_{11}, B_{12}, \ldots)$ might represent purchasers. A_1 is the purchaser ID, the key for this relation. The B_{1i}'s might give the purchaser's name, address, phone, and so on. Typically, the fact table is much larger than the dimension tables. For instance, there might be a fact table of a billion tuples and ten dimension tables of a million tuples each.

Analysts mine this data by asking *analytic queries* that typically join the fact table with several of the dimension tables (a "star join") and then aggregate the result into a useful form. For instance, an analyst might ask "give me a table of sales of pants, broken down by region and color, for each month of 2012." Under the communication-cost model of this section, joining the fact table and dimension tables by a multiway join is almost certain to be more efficient than joining the relations in pairs. In fact, it may make sense to store the fact table over however many compute nodes are available, and replicate the dimension tables permanently in exactly the same way as we would replicate them should we take the join of the fact table and all the dimension tables. In this special case, only the key attributes (the A's above) are hashed to buckets, and the number of buckets for each key attribute is inversely proportional to the size of its dimension table.

Naturally, $a_1 a_2 \cdots a_m = k$. Finally, suppose each dimension table D_i has size d_i, and the size of the fact table is much larger than any of these sizes. Find the values of the a_i's that minimize the cost of taking the star join as one map-reduce operation.