

12 Outline

- Map reduce and indexing
- Sparse matrix multiplication using SQL
- Joins using map reduce

12.1 Map Reduce

Recall what map reduce does: it's a tool for putting pairs that have the same key together. The map reduce environment gave the following promise:

- Map phase: emits pairs of the form $\langle \text{key}, \text{value} \rangle$ (shuffle).
- Shuffle phase: places pairs with the same key on a single machine.
- Reduce phase: performs computations on pairs with the same key.

On the face of it, this may look trivial. However, this construct is powerful just like a `for` loop. If you write your code according to this contract you get fault tolerance, you get distribution, and utilization of all cores in the cluster.

Example Suppose we've indexed the web by crawling it with a thousand machines. So now each machine has stored on it a part of the web; each machine is full of HTML and pointers to other HTML pages. From this, we'd like to build a search engine kind of like Google. So, how can we apply map reduce to a search engine? Given a key word, we need to be able to *very* quickly figure out which web-pages have that key in them; this is the most basic search engine functionality. If we have an inverted-index, we can do this efficiently.

An inverted-index is a list of words, where each word is associated with the pages it occurs in. So, we'd like to get to a point where we have a list

$$\text{word}_i \rightarrow \{\text{page}_1, \text{page}_2, \dots\}$$

How can we create such an index using map reduce? All we have is a jumble of web-pages, and we want a map between the word itself and the pages on which it occurs. We only have one hammer at our disposal, map-reduce, which puts pairs with the same key together. So, naturally, the key should be the word itself, and the value should be the url.

```

// Map phase
1 for  $w \in H$  do
2   | emit ( $w, h_{uid}$ )
3 end
// Reduce phase
4 Reduce( $w, \langle url_1, url_2, \dots, \rangle$ )

```

Algorithm 1: Mapper

So, each mapper will examine the web-page that it has been given. For each word in the web-page, emit the word as the key and the url as the value.

The above code is low-level, and can be tedious to write ourselves. Instead, we move onto SQL, and specifically multiplying sparse matrices.

12.2 Sparse Matrix-Vector Multiply using SQL

First of all, let's figure out how our matrix is represented. Since our matrix M is sparse, we need the following representation

$$(i, j, \text{value})$$

i.e. for each non-zero entry we store the row and column indices alongside the corresponding value. A sparse vector x is represented as (i, value) .

Now, imagine that our matrix is an SQL table, where the columns in the table is i , j , and the value; similarly for our matrix, represented as a table with a column for the i index and a column for the value.

How can we compute Mx using SQL operations?¹ We realize that if $Mx = b$, then each entry of b an inner product between corresponding row of M and vector x . But how can we prepare these dot-products? We first perform a join on M and x where we join M on its column index j and x on its element index i . Our resulting table looks like

$$\overline{\overline{M.j(x_i) \quad M.i \quad M.value \quad x.value}}$$

Notice that where there are duplicates in our join, a cartesian product will appear. I.e. for every single entry in matrix M , we will get the corresponding value from vector x .

$$\begin{array}{r}
 M = \begin{array}{c|cc}
 i & j & \text{value} \\
 \hline
 1 & 2 & 17.5 \\
 2 & 2 & 16.3
 \end{array} \\
 \\
 x = \begin{array}{c|c}
 i & \text{value} \\
 \hline
 1 & 2.71 \\
 2 & 3.14
 \end{array}
 \end{array}$$

¹First attempt: We can first group by i on matrix table M ; at this point, each entry in the resulting table will correspond to a row in M , where the value is a function of the values in the columns of M for said row. When we do a **group-by**, we need to specify what kind of aggregation function we perform to combine the values for a particular group. What kind of aggregation should we use? We abandon this approach.

The resulting table after our `inner-join`

$M.j(x_i)$	$M.i$	$M.value$	$x.value$
2	1	37.5	3.14
2	2	18.3	3.14

The second SQL statement simply adds a new column onto this table. The column will look like

$M.j(x_i)$	$M.i$	$M.value$	$x.value$	mult
2	1	37.5	3.14	37.5×3.14
2	2	18.3	3.14	...

We've done the multiplication in each dot-product. Now we need to perform the summation, i.e. perform a group by i and sum over our column `mult` which we added above. We'd like to take all of the multiplications that are associated with a particular dot-product and sum them up, so our aggregation function is a summation on `mult`. Notice that we only need as output the index i and corresponding value of Mx , so we can remove other columns.

Now that we've done this in SQL, how can we do this in map-reduce?

12.3 SQL Group-by using map reduce

We first implement a `group-by`, which groups all rows that have the same key and performs an aggregation function on remaining columns. A map reduce is just that. To implement a `group-by` using map reduce, all we have to do is

- 1 emit the group-by column as key
- 2 perform desired aggregation function on non-group-by columns in the reducer

Algorithm 2: group by using map reduce

Let's walk through an example. Suppose we have the following table.

a	b
1	1
1	2
3	3
4	5
5	11

We wish to `group-by` a and sum elements in column b . How can we do this in map reduce? Each row assumed to be an input to a mapper.

```
Group-by a
  sum(b)
```

So, a map reduce is essentially a group-by; they're effectively the same.

12.4 SQL (inner) join using map reduce

This is much more difficult than a `group-by`. There are several cases:

- both tables are so large that neither will fit on a single machine, and
- one table is small enough to fit locally.²

The implementations for these two cases are very distinct. In our join, we need to respect the duplicity of the keys. I.e. we must perform the cross-product with all entries appearing in the other table with the same key.

The case when one table fits in memory Suppose we want to merge tables T_1 and T_2 , where table T_1 is small enough to fit in memory. For the sake of discussion, let's suppose that each table has two columns.

$$T_1 = \overline{\begin{array}{cc} a & b \end{array}}, \quad T_2 = \overline{\begin{array}{cc} i & j \end{array}}$$

Do we even need a cluster to perform this? Yes, we definitely need a distributed environment, since the resulting join could be larger than table B . But do we need to perform our sorting operation on our tables? No, we will see that we actually don't need this.

We first broadcast T_1 to all machines via a bit-torrent broadcast. Now, each machine has a full-on copy of table T_1 . For fast look-up, we place T_1 into a hash-table.

- 1 Place T_1 in a hash-table for fast look-up
- 2 Place hashed T_1 on each machine via a broadcast
- 3 Perform an inner join of whatever data in T_2 is stored locally on each machine with its local copy of (hashed) T_1

Algorithm 3: (Hash or Broadcast) Inner Join

Each machine does a join of what it owns from T_2 with *all-of* T_1 . We don't even need to communicate this result to other machines, we can just leave the merged data local. The inner join that is performed locally on each machine can be done just as we did on the midterm. We assume that the resulting table can be fit on each machine; in the event that we have a cross-product which blows up our table, we would just start assuming that T_1 so large it can't fit on a single machine. We cover this case next, i.e. the case where neither table can fit locally.

References

- [1] R. Zadeh. *Introduction to Distributed Optimization*.
- [2] R. Zadeh. *Distributed Computing with Spark and MapReduce*.

²A third case is actually that both tables can fit on a single machine. This is covered by our PRAM model, and we saw this exact question on the midterm.