MapReduce examples

CSE 344 — section 8 worksheet

May 19, 2011

In today's section, we will be covering some more examples of using MapReduce to implement relational queries. Recall how MapReduce works from the programmer's perspective:

1. The input is a set of (key, value) pairs.
2. The map function is run on each (key, value) pair, producing a bag of intermediate (key, value) pairs:

   ```
   map (inkey, invalue):
     // do some processing on (inkey, invalue)
     emit_intermediate(hkey1, hvalue1)
     emit_intermediate(hkey2, hvalue2)
     // ...
   ```

3. The MapReduce implementation groups the intermediate (key, value) pairs by the intermediate key. Despite the name, this grouping is very different from the grouping operator of the relational algebra, or the `GROUP BY` clause of SQL. Instead of producing only the grouping key and the aggregate values, if any, MapReduce grouping also outputs a bag containing all the values associated with each value of the grouping key. In addition, grouping is separated from aggregation computation, which goes in the reduce function.
4. The reduce function is run on each distinct intermediate key, along with a bag of all the values associated with that key. It produces a bag of final values:

   ```
   reduce(hkey, hvalues[]):
     // do some processing on hkey, each element of hvalues[]
     emit(fvalue1)
     emit(fvalue2)
     // ...
   ```

Suppose you have two relations: $R(a, b)$ and $S(b, c)$, whose contents are stored as files on disk. Before you can process these relations using MapReduce, you need to parse each tuple in each relation as a (key, value) *pair*.

This task is relatively simple for our two relations, which only have two attributes to begin with. Let's treat $R.a$ as the "key" for tuples in R (note that this does not have to be a true key in the relational sense), and similarly, let's treat $S.b$ as the key for S.

For the "values" of the (key, value) pairs, we won't use the single non-key attribute. Instead, we'll use a composite value consisting of both attributes, plus a tag indicating where the value came from. So, for relation R, the values will have three components, value.tag, which is always the string "R", and value.a and value.b for the two attributes of R. Similarly, relation S's tuples will have MapReduce values containing value.tag, which is always "S", and value.b and value.c for S's attributes. This convention allows us to use the same `map` function for tuples from both relations; we'll see the use of that soon.

Given this notation for the (K, V) pairs of R and S, let's try to write pseudocode algorithms for the following relational operations:

1. Selecting tuples from R: $\sigma_{a<10}R$

2. Eliminate duplicates from R: $\delta(R)$

3. Natural join of R and S: $R \bowtie_{R.b=S.b} S$

4. 3-way natural join: $R \bowtie_{R.b=S.b} S \bowtie_{S.c=T.c} T$, where we introduce a new relation $T(c, d)$.

5. Grouped and aggregated join: $\gamma_{(a,\text{sum}(c)\to s)}\,(R \bowtie_{R.b=S.b} S)$