Mining Massive Datasets: Week 1: MapReduce

Ian Quah (itq)

June 9, 2017

Topic 1

Map, Reduce and program structure

1. Map: (input elements) → (0 ++ key-value pairs)
2. Reduce (key [V_1, V_2, ... , V_n]) → (0 ++ key-value pairs)
   - (a) If the reducer function is associative and commutative (order doesn't matter) can push off computation to map function.
   - (b) This is the concept behind a combiner: (map → reduce) → reduce

     Examples:
     - i. sum is associative and commutative
     - ii. average is not, however, if we store the aggregate as (sum, # elems), we can reduce the number of tuples required to be transferred
3. Functions provided by the user:
   - (a) map
   - (b) reduce
   - (c) hash (optional): for user-specified grouping + specifying where to send jobs to
4. Structure of Program
   - (a) Master node sends data to all compute nodes, then periodically pings them
   - (b) Dealing with Failure:
     - i. Only failure on master can bring the whole thing down
     - ii. Bc master can know when things are down (ping), it can detect if a node failed if it did, the entire calculation must be restarted (as data for reduce was stored in failed node)
     - iii. Master assigns the original task to whichever cluster frees up, then notifies reduce that the source of incoming data is changing

Topic 2

Relational algebra

Topic 2 continued on next page. . .

1. Definitions:

   - Attributes: Column headers
   - Relation: table with attributes
   - tuples: rows of the relation
   - Schema: set of attributes of a relation
   - R(A_1, A_2, ..., A_n): - Relation name: R
     - attributes: A_1,....A_n

1. Relational Algebra Operations and definitions

   - Selection:
     - Apply condition C to each tuple.
     - Returns only tuples that satisfy C
     - Denoted σ_C(R)
   - Projection:
     - For some subset S of attributes ofR
     - produce from each tuple only components for attributes in S
     - Denoted π_S(R)
   - Union, Intersection, Difference:
     - Duh
   - Natural Join:
     - Denoted as $R_1 \bowtie R_2$
     - Given R_1, R_2, compare each pair of tuples
     - If agree on ALL attributes common between the two:
       - return tuple s.t
       - **Cond 1**: has components for each attribute in either schema
       - **Cond 2**: agrees with two tuples on all attribs
     - else:
       - return None
     - Discussion below talks about Nat Joins, but also applies to *equijoins* - equality covers equality of attributes that do not necessarily have the same name
   - Grouping and Aggregation:
     - Duh