Exam 2021

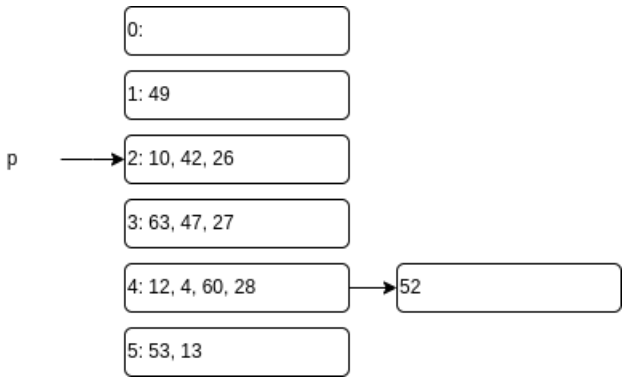
June 17, 2023

Exercise 0.1. Given N replicas, let's call R the ReadConcern parameter of MongoDB and W the WriteConcern (which indicate respectively the number of copies it reads and writes, before confirming the operation to the user); give the equation involving those variables that corresponds to the eventually consistent configuration.

Eventual consistency is achieved when

$$W + R \leq N.$$

Exercise 0.2. Given an empty linear hash with f(x) = x (i.e., we directly apply the module to the keys), and a capacity of four keys per bucket, draw the result of inserting the following keys in the given order: 12, 4, 10, 49, 42, 60, 63, 53, 47, 27, 26, 28, 13, 52.



Exercise 0.3. Given a file of 3.2GB stored in an HDFS cluster of 50 machines, and containing $16 \cdot 10^5$ key-value pairs in a SequenceFile; estimate the execution time of a Spark job containing a single map transformation and an action storing the results in a file. Explicit any assumption you make and consider also the following parameters:

- Chunk size: 128MB (default)
- Replication factor: 3 (default)
- Map function (i.e., the parameter of the transformation) execution time: 10^{-3} sec/call (this is the only cost you have to consider)
- Save action execution time: 0sec (do not consider its cost at all)

The number of chunks is

$$N = \frac{3.2GB}{128MB} = 25.$$

Therefore, there are

$$N_{kv} = \frac{16 \cdot 10^5}{25} = 64000$$

key-value pairs per chunk. Since there are 50 machines, all chunks can be processed in parallel, accounting for a total execution time of

$$T = t \cdot N_{kv} = 10^{-3} \cdot 64000 = 64s.$$

Exercise 0.4. Analyze (i.e., briefly give pros and cons) the following JSON design compared to other equivalent JSON designs from the three perspectives:

```
" BID - PRODUCT ":{
  " B_ID ": int (4), " B_PRICE ": int (4), " U_ID ": int (4),
  " PRODUCT ": {
    " P_ID ": int (4), " P_INFO ": varchar (100)
}}
  " PRODUCT - SELLER - REGION ": {
  " P_ID ": int (4), " P_INFO ": varchar (100),
  " USER ": {
    " U_ID ": int (4), " U_F_NAME ": varchar (20),
    " REGION ": {
      " R_ID ": int (4), " R_NAME ": varchar (10)
}}}
" PRODUCT - COMMENTS ": {
  " P_ID ": int (4), " P_INFO ": varchar (100),
  " COMMENTS ": [{
    " C_ID ": int (4), " C_TITLE ": varchar (20), " U_ID ": int (4)
}]}
```

- Read (a.k.a. Query):
  - Pros: retrieving the information regarding a product is very easy, like the sellers of a product. Also, retrieving all the comments of a product is easy.
  - Cons: retrieving all products that a seller offers requires many joins, as well as obtaining all bids to a product and even worse is to find all products sold in a given region.
- Update:
  - Pros: if we want to update the sellers of a particular product, it is fast and easy. Same if we want to update the comments of a particular product or the product of a bid.
  - Cons: if we want to update the region of a seller, we would need to traverse all product to see if the seller sells that product, and then update the region. This is very costly and inefficient. Also, if we want to update a product description, we would need to change it in the three places that it appears.
- Memory:
  - Pros: Having product as a central concept makes BID-PRODUCT and PRODUCT-COMMENTS able to be stored efficiently, since we just need to provide the ID of the related product.
  - Cons: P_INFO is stored repeatedly. Not only this, but we are storing the USERS many times, and the REGIONS even more times.

Exercise 0.5. Assume we have a MongoDB collection in a distributed cluster, which contains prices of apartments without any secondary index. Such collection is big enough not to completely fit in memory. We want to use Spark to compute the standard deviation per neighbourhood. Clearly identify the most efficient option and briefly justify the choice (it is not necessary to provide the Spark code).

A. Use Spark only to push the query to MongoDB aggregation framework and simply get the result. No, because we would need the mongo router to perform all the work, losing the power of parallelism.

B. Push only some of the operations to MongoDB aggregation framework and the run the rest in Spark. Yes, but we need specific drivers for Spark to be able to push operations to the storage engine. For MongoDB these exist.

C. Load the whole collection to an RDD and perform all computations in Spark. No, because we would need to transfer everything to Spark, implying a huge data transfer overload.