Exercises Big Data Management

Database Technologies and Information Management (DTIM) group
Universitat Politècnica de Catalunya (BarcelonaTech), Barcelona
April 29, 2025

Contents

1 Document Stores 5
1.1 Problems 5

1 Document Stores

1.1 Problems

1. Optimize the performance of the following document design:

```
{"_id": 123,
 "name": "Alberto"
 "address": {
  "street": "Jordi Girona",
  "city": "Barcelona",
  "zip_code": "08034"},
 "telephone1": "93 4137889",
 "telephone2": "93 4137840",
 "telephone3": "123456789"
}
```

2. 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)
}]}
```

   (a) Read (a.k.a. Query)
      i. Pros:
      ii. Cons:
   (b) Update
      i. Pros:
      ii. Cons:
   (c) Memory (a.k.a. Space)
      i. Pros:
      ii. Cons:

3. Optimize the performance of the following pipe:

```
db.partsupp.aggregate([
 {"$sort": {"n_name": 1, "s_name": 1, "p_partkey": 1}},
 {"$match": {"part.p_size": 5},
 {"$project": {
   "supplier.s_name": 1,
   "supplier.s_phone": 1,
   "supplier.s_comment": 1,
   "_id": 0
   }}
])
```

4. Assume a MongoDB collection of JSON documents following the structure shown below. Indicate a sequence of stage operators that can be used in a pipeline to compute the top five departments in terms of number of employees such that its location has at least one `performance_review` greater than 7. In the example below, since both locations have performance reviews greater than 7, `manufacturing` has a total of 90 employees, while `billing` and `sales` have 80 and 40 employees, respectively. You need only provide the names of the stage operators and the sequence in which they are applied. You should not use the `mapreduce` feature of MongoDB but rather the specific stage operators provided by MongoDB. You do not need to explicitly handle the case of ties (i.e., in the case of ties, you may return any of the tied documents to complete the top five).

```
{ "location": "Barcelona",
  "budget": 3.000.000,
  "departments": [ { "name": "sales", "employees": 40 }, { "name": "manufacturing",
  "employees": 75 } ],
  "performance_reviews": [ 7, 6, 3 ]
},
{ "location": "Brussels",
  "budget": 5.000.000,
  "departments": [ { "name": "manufacturing", "employees": 15 }, { "name": "billing
  ", "employees": 80 } ],
  "performance_reviews": [ 6, 8, 7 ]
}
```

5. Imagine you want to design a system to maintain data about citizens and doubt whether to use HBase or MongoDB. Precisely, for each citizen, it will store: their personal data (with pID), their city data (with cID) and their employment data. We also know the workload (i.e., queries and frequency of execution) is as follows:

   - Q_1: Average salary in Barcelona (50% frequency) – information obtained from the set of city and employment data.
   - Q_2: Average weight for young people (less than 18 years old) (45% frequency) – information obtained from the set of personal data.
   - Q_3: Number of VIPs (5% frequency) – information obtained from the set of personal data.

   ⇒ Discuss your choice of technology and data model, without pre-computing the results of the queries. Clearly specify the structure of the data (i.e., tables/collections, keys, values, etc.), trade-offs and assumptions made.

6. Assume you have a MongoDB collection which occupies 6 chunks **evenly distributed** in 3 shards (i.e., 2 chunks per shard). Being the document ID also the shard key, the chunk of a document is determined **by**

means of a hash function. Assuming that accessing one document takes one time unit (existing indexes are used at no cost) and we have 6,000 documents in the collection, k of which have value "YYY" for attribute "other", how many time units would take the following operations[1]:

a) $FindOne(\{\_id : "XXX"\})$

b) $Find(\{\_id : \{\$in : [1, .., 3000]\}\})$, being [1,..,6000] the range of existing IDs.

c) $Find(\{other : "YYY"\})$, being the attribute indexed.

d) $Find(\{other : "YYY"\})$, being the attribute NOT indexed.

7. Assume you have a MongoDB collection which occupies 6 chunks **UNevenly distributed** in 3 shards (i.e., 1, 2 and 3 chunks per shard respectively). Being the document Id also the shard key, the chunk of a document is determined **by means of a hash function**. Assuming that accessing one document takes one time unit (existing indexes are used at no cost) and we have 6,000 documents in the collection, k of which have value "YYY" for attribute "other", how many time units would take the following operations[2]:

   e) $FindOne(\{\_id : "XXX"\})$

   f) $Find(\{\_id : \{\$in : [1, .., 3000]\}\})$, being [1,..,6000] the range of existing IDs.

   g) $Find(\{other : "YYY"\})$, being the attribute indexed.

   h) $Find(\{other : "YYY"\})$, being the attribute NOT indexed.

8. Assume you have a MongoDB collection which occupies 6 chunks and is **evenly distributed** in 3 shards (i.e., 2 chunks per shard) . Being the document Id also the shard key, the chunk of a document is determined **by range**. Assuming that accessing one document takes one time unit (existing indexes are used at no cost) and we have 6,000 documents in the collection, k of which have value "YYY" for attribute "other", how many time units would take the following operations[3]:

   i) $FindOne(\{\_id : "XXX"\})$

   j) $Find(\{\_id : \{\$in : [1, .., 3000]\}\})$, being [1,..,6000] the range of existing IDs.

   k) $Find(\{other : "YYY"\})$, being the attribute indexed.

   l) $Find(\{other : "YYY"\})$, being the attribute NOT indexed.

[1] As typically in RDBMS optimizers, assume uniform distribution of values and statistical independence between pairs of attributes.
[2] As typically in RDBMS optimizers, assume uniform distribution of values and statistical independence between pairs of attributes.
[3] As typically in RDBMS optimizers, assume uniform distribution of values and statistical independence between pairs of attributes.