Designing Applications

Source: MongoDB: The Definitive Guide, Second Edition

Indexing

Source: MongoDB: The Definitive Guide, Second Edition

Index

- Similar to a book index
- An ordered list that points to content
  - Allows it to query orders of magnitude faster
- Other approach is “table scan”

Source: MongoDB: The Definitive Guide, Second Edition

Task

- Create a PCSMA database and users collection and insert 10 million documents in it with fields of
  - "i" : i,
  - "username" : "user"+i,
  - "age" : Math.floor(Math.random()*120),
  - "created" : new Date()

- Do a query on it for “user101”

Source: MongoDB: The Definitive Guide, Second Edition

Task

- db.users.find({username: "user101"}).explain(“executionStats”)
- db.users.find({username: "user101"}).limit(1).explain(“executionStats”)

Source: MongoDB: The Definitive Guide, Second Edition

Indexing

- Now create an Index on username
  - db.users.ensureIndex({"username" : 1})
- Do the search again

Source: MongoDB: The Definitive Guide, Second Edition

Cost of indexing

- MongoDB has to update all your indexes whenever your data changes, as well as the document itself
  - every write (insert, update, or delete) will take longer for every index you add
- MongoDB limits 64 indexes per collection

Source: MongoDB: The Definitive Guide, Second Edition

How to choose write fields

- Look through your common queries
- Look for queries that need to be fast
- Try to find a common set of keys from those

Source: MongoDB: The Definitive Guide, Second Edition

Compound Indexes

- db.users.find().sort({"age" : 1, "username" : 1})
- db.users.ensureIndex({"age" : 1, "username" : 1})

- db.users.find({"age" : 21}).sort({"username" : -1})
- db.users.find({"age" : {"$gte" : 21, "$lte" : 30}})
- db.users.find({"age" : {"$gte" : 21, "$lte" : 30}}).sort({"username" :1})

```
[0, "user100309"] -> 0x0c965148
[0, "user100334"] -> 0xf51f818e
[0, "user100479"] -> 0x00fd7934
...
[0, "user99985" ] -> 0xd246648f
[1, "user100156"] -> 0xf78d5bdd
[1, "user100187"] -> 0x68ab28bd
[1, "user100192"] -> 0x5c7fb621
...
[1, "user999920"] -> 0x67ded4b7
[2, "user100141"] -> 0x3996dd46
[2, "user100149"] -> 0xfce68412
[2, "user100223"] -> 0x91106e23
```

Source: MongoDB: The Definitive Guide, Second Edition

Compound Indexes

- What if the sort is in this order
  - {"username" : 1, "age" : 1}

- db.users.find({"age" : {"$gte" : 21, "$lte" : 30}}).sort({"username" :1})

```
["user0", 69]
["user1", 50]
["user10", 80]
["user100", 48]
["user1000", 111]
["user10000", 98]
["user100000", 21] -> 0x73f0b48d
["user100001", 60]
["user100002", 82]
["user100003", 27] -> 0x0078f55f
["user100004", 22] -> 0x5f0d3088
["user100005", 95]
...
```

Source: MongoDB: The Definitive Guide, Second Edition

Compound Indexes

- Choosing key directions
  - Having Index keys going in different directions
- Covered Indexes
  - When the index itself covers the required field

Source: MongoDB: The Definitive Guide, Second Edition

$-operator and Indexes

- Some queries cannot use indexes
  - $exists
  - $where
  - $not
  - …
- $or allows to use more than one index
  - db.foo.find({"$or" : [{"x" : 123}, {"y" : 456}]})

Source: MongoDB: The Definitive Guide, Second Edition

Other points

- When designing an index with multiple fields, put fields that will be used in exact matches first (e.g., "x" : "foo") and ranges last (e.g., "y": {"$gt" : 3, "$lt" : 5}).
- Indexes can be created on keys in embedded documents in the same way that they are created on normal keys

```
{
        "username" : "sid",
        "loc" : {
                "ip" : "1.2.3.4",
                "city" : "Springfield",
                "state" : "NY"
        }
}
```

  - db.users.ensureIndex({"loc.city" : 1})
- Greater the cardinality of the field, more helpful is the index on that field

Source: MongoDB: The Definitive Guide, Second Edition

Query Optimizer

- If an index exactly matches a query (you are querying for "x" and have an index on "x"), the query optimizer will use that index.
- Otherwise, MongoDB will select a subset of likely indexes and run the query once with each plan, in parallel.
- The first plan to return 100 results is the “winner” and the other plans’ executions are halted.
- This plan is cached and used subsequently for that query until the collection has seen a certain amount of churn.
- Once the collection has changed a certain amount since the initial plan evaluation, the query optimizer will re-race the possible plans.
- Plans will also be reevaluated after index creation or every 1,000 queries.
- Use hint() to suggest an index

Source: MongoDB: The Definitive Guide, Second Edition

When Not to Index

- Indexes are most effective at retrieving small subsets of data
- Indexes become less and less efficient as you need to get larger percentages of a collection
  - Using an index requires two lookups

- No hard-and-fast rules
  - if a query is returning 30% or more of the collection, start looking at whether indexes or table scans are faster.

Source: MongoDB: The Definitive Guide, Second Edition