MongoDB and CouchDB: A Brief Comparison

IANNETTA Paul

October 13, 2017

Abstract

This report aims at explaining briefly what are document related databases, what operations and guarantees they provide. In a second time, a short comparison about how to create a database, add documents in it, update them and query the database will be done with the two following databases: MongoDB and CouchDB.

1 Document Oriented Databases

Document oriented databases are one of the main categories of NoSQL databases and a subcategory of key-value database. Indeed, in this case, the value is a document. A document is a semi-structured kind of data which can be anything from raw text to XML, JSON or YAML. The main advantages of document-oriented databases over more general key-value databases is that since the format of the document is known in advance the database management process can harness its features so that it performs better in most cases than a general key-value database. MongoDB (1st) and CouchDB (4th) are among the most popular document-oriented database1.

A document-oriented database provide the following instructions: CRUD (Creation, Retrieval, Update, Deletion) at least.

1.1 MongoDB

The increasingly2 most popular document-oriented database: MongoDB, is an open source project (written mainly in C++) under the supervision of the Mongo Inc. The source code is licensed under both the GNU AGPL v3.03 and the Apache License4. The documents in MongoDB are stored as JSON. The main interface with MongoDB is through its shell API. However, MongoDB has a certain number of drivers in a wide variety of languages including Ruby, PHP and Python.

1.2 CouchDB

This lightweight albeit very popular database system is an open source project (written mainly in Erlang—a functional concurrent language) maintained by the Apache Foundation and licensed under the Apache License 2.05. CouchDB API also uses HTTP requests and serves JSON documents over a REST/HTTP. Therefore a driver for CouchDB can be written as long as the language support HTTP requests.

2 The ACID criterion

Atomicity, Consistency, Isolation, Durability, is a set of properties of database transactions designed to prevent errors and maintain the validity of a database no matter what may happen (network errors, etc.) Both CouchDB and MongoDB implement the ACID criterion.

Atomicity forces that all transactions are either all or nothing. This policy enforces that a partially failed transaction has no effect on the database. For example, in the case of an online purchase, if the database suffers an error it is desirable that the transaction is fully canceled rather than either the company having the money and the customer order not being registered or vice versa.

Consistency is the fact that each transaction will bring the database from a valid state to another valid state.

Isolation is the fact that all transactions can safely being executed concurrently and will have the same result has if they were executed sequentially.
Durability is the fact that no matter what may happen each fully completed transaction’s result is stored permanently (in a non volatile memory).

2.1 CRUD

The acronym CRUD refers to the main operations available on databases: Creation, Read, Update and Deletion. Those are available for both CouchDB and MongoDB through their RESTful HTTP API (in bold the request types actually used but CouchDB and MongoDB).

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Request equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>PUT / POST</td>
</tr>
<tr>
<td>Read</td>
<td>GET / HEAD</td>
</tr>
<tr>
<td>Update</td>
<td>PATCH / POST / PUT</td>
</tr>
<tr>
<td>Delete</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

3 Personal Experience

3.1 Installation

Conveniently, Gentoo provides an ebuild for both MongoDB and CouchDB. It should be noted that MongoDB takes much more time compile than CouchDB. On my machine (RAM: 6Gb, Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz), CouchDB takes around 30 minutes to compile while CouchDB needs only 2 minutes.

Both are easily started with the following commands (executed as root or with enough priviledges):

```
service mongod start
service couchdb start
```

With no further configuration, MongoDB starts listening on port 27017, while CouchDB starts listening on port 5984.

3.2 Testing

As my main database I will use a file that contains a dump of the characters used in Japan as well as how they are read. The dump contain a little more than 2000 entries with 8 fields each.

In order to create the database for CouchDB:

```
$ mongoimport --jsonArray --db kanji --collection docs --file jouyou.json
```

```
$ mongo
> show dbs
admin 0.000GB
kanji 0.000GB
local 0.000GB
> use kanji
switched to db kanji
```

# Let's find character with a specific radical.

```
> db.docs.find({"radical": "⺨"})
```

```
{ "_id" : ObjectId("59dfea970d7cc725b5b8365b"), "kanji" : "犬", "radical" : [ "犬", "⺨" ], "rad_names" : [ "いぬ", "けものへん" ], "shou" : [ "ケン", "いぬ" ], "chuu" : [ ], "kou" : [ ], "gai" : [ ], "all" : [ "ケン", "いぬ" ] }
```

# Now with a fixed reading.

```
> db.docs.find({"all": "みことのり"})
```

```
{ "_id" : ObjectId("59dfea970d7cc725b5b836dd"), "kanji" : "犯", "radical" : [ "犬", "⺨" ], "rad_names" : [ "いぬ", "けものへん" ], "shou" : [ "ハン" ], "chuu" : [ "おかす" ], "kou" : [ ], "gai" : [ "ポン" ], "all" : [ "ハン", "おかす", "ポン" ] }
```

"..."
Let's now do the same with COUCHDB.

Let's update a document.

> db.dropDatabase()
> show dbs
local 0.000GB
> exit
bye

From there we need to use COUCHDB views in order to query the database.

We will use the following view:

```
{ "_id": "_design/radical",
  "language": "javascript",
  "views": {
    "kemonohen": { "map": "function(doc) { if (doc.radical.indexOf('矛') != -1) emit(doc) }" }
  }
}
```

Then we add the view and use it:

```
curl -X GET localhost:5984/kanji/_design/radical
```

```
curl -H "Content-Type: application/json" -d @jouyou_couch.json -X POST localhost:5984/kanji/_bulk_docs
```

```
$ curl -X PUT localhost:5984/kanji/"ok":true
```

```
$ curl -H "Content-Type: application/json" -d @radical.json localhost:5984/kanji/_design/radical/"ok":true,"id":_design/radical,"rev":1-406ea9251992323279cdeb4af87eb7e7
```

```
curl -X GET localhost:5984/kanji/_design/radical/_view/kemonohen
```

```
{ "id": "1213", "key": "1213", "_rev": "1-c5441243374475da3d624a80a94df1fc", "kanji": "猛", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["モウ", "モウ", "モウ"], "kou": [], "gai": ["たけし"], "all": ["モウ", "たけし"], "value":null},
```

```
{ "id": "1246", "key": "1246", "_rev": "1-c8cc32a63cbfc310f3a6645389e9ce65d3", "kanji": " целью", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["リョウ", "かり", "かかる"], "all": ["リョウ", "かり", "かかる"], "value":null},
```

# Let's update a document with `save` rather than `update` to make the change persistent.

> db.docs.save({"_id": ObjectId("59dfea3f0d7cc725b5b83388"), "to_be_deleted":"true"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

# The change is effective, indeed.

> db.docs.find({"to_be_deleted":true})
```
{ "_id": ObjectId("59dfea3f0d7cc725b5b83388"), "to_be_deleted": "true" }
```

> db.docs.find()
```
{ "_id": ObjectId("59dfea3f0d7cc725b5b83388"), "kanji": "けものへん", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["モウ", "モウ", "モウ"], "kou": [], "gai": ["たけし"], "all": ["モウ", "たけし"], "value":null},
```

Let's update a document with `save` rather than `update` to make the change.

```
$ curl -X PUT localhost:5984/kanji
```

```
$ curl -H "Content-Type: application/json" -d @jouyou_couch.json -X POST localhost:5984/kanji/_bulk_docs
```

```
{ "_id": ObjectId("59dfea3f0d7cc725b5b83388"), "to_be_deleted": "true" }
```

```
$ curl -H "Content-Type: application/json" -d @radical.json localhost:5984/kanji/_design/radical
```

```
{ "_id": ObjectId("59dfea3f0d7cc725b5b83388"), "kanji": "けものへん", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["モウ", "モウ", "モウ"], "kou": [], "gai": ["たけし"], "all": ["モウ", "たけし"], "value":null},
```

We can see that in both cases the `find` request returns all entries containing the requested string in the requested field.

Now, let's update a document.

We will use the following view:

```
{ "id": ObjectId("59dfea3f0d7cc725b5b83388"), "kanji": "けものへん", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["モウ", "モウ", "モウ"], "kou": [], "gai": ["たけし"], "all": ["モウ", "たけし"], "value":null},
```

Let's update a document with `save` rather than `update` to make the change persistent.

```
> db.dropDatabase()
> show dbs
local 0.000GB
> exit
bye
```

We can see that in both cases the `find` request returns all entries containing the requested string in the requested field.

Now, let's update a document.

We will use the following view:

```
{ "id": ObjectId("59dfea3f0d7cc725b5b83388"), "kanji": "けものへん", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["モウ", "モウ", "モウ"], "kou": [], "gai": ["たけし"], "all": ["モウ", "たけし"], "value":null},
```

We can see that in both cases the `find` request returns all entries containing the requested string in the requested field.

Now, let's update a document.

We will use the following view:

```
{ "id": ObjectId("59dfea3f0d7cc725b5b83388"), "kanji": "けものへん", "radical": ["犬", "犬"], "rad_names": ["いぬ", "けものへん"], "shou": [], "chuu": ["モウ", "モウ", "モウ"], "kou": [], "gai": ["たけし"], "all": ["モウ", "たけし"], "value":null},
```
We now use a new view to support the querying of characters read as "みことのり".

```json
{
   "id": "_design/all",
   "language": "javascript",
   "views": {
      "mikotonori": { "map": "function(doc) { if (doc.all.indexOf('みことのり') != -1)
                   emit(doc) }"}
   }
}
```

and we use it:

```bash
curl -H "application/json" -X PUT -d @radical.json localhost:5984/kanji/_design/all
"ok":true,"id": "_design/all","rev": "1-85d220e14e97d5f564179918d09bf510"
curl -X GET localhost:5984/kanji/_design/all/_view/mikotonori
{"total_rows":2,"offset":0,"rows": ["id":"1487","key":{_id:"1487"},"_rev": "1-7c0c287d33a9483080f54072f603ed8","kanji":
   "詔","radical": ["言"],"rad_names": ["もんべん","ことば"],"shou": [],"chuu": ["ショウ"],"kou": ["みことのり"],"gai": ["つげる"],"all": ["ショウ","みことのり","つげる"],"value": null},
{"id":"859","key":{_id:"859"},"_rev": "1-839f73571e5aa3a2f5ad18c7f60273d3","kanji":
   "勅","radical": ["力"],"rad_names": ["ちから"],"shou": [],"chuu": ["チョ"],"kou": [],"gai": ["いましめる","みことのり"],"all": ["チョク","いましめる","みことのり"]},"value": null}
```

```bash
# Again let's get rid of 詔

curl -X DELETE localhost:5984/kanji/1487?_rev=1-7c0c287d33a9483080f54072f603ed8
"ok":true,"id": "1487","rev": "2-10958fad3a1c2310495ad18c7f8381fc"
curl -X GET localhost:5984/kanji/_design/all/_view/mikotonori
{"total_rows":1,"offset":0,"rows": ["id":"859","key":{_id:"859"},"_rev": "1-839f73571e5aa3a2f5ad18c7f60273d3","kanji":
   "勅","radical": ["力"],"rad_names": ["ちから"],"shou": [],"chuu": ["チョ"],"kou": [],"gai": ["いましめる","みことのり"],"all": ["チョク","いましめる","みことのり"]},"value": null}
```

```bash
# Everything is fine
# Let's now delete the database

curl -X DELETE localhost:5984/kanji/
"ok":true
```

4 Conclusion

I did not mention how those databases handles the ACID properties because it is difficult to test on a single computer nevertheless both system implements them with a fine grain so that distributed accesses are well handled and those systems scale well horizontally.

From a personal point of view, MONGOCLDB felt easier to use because it could handle requests directly without any need to fiddle with any map/reduce system. Nevertheless, the approach, taken by COUCHDB, of using a map/reduce system called views in order to query the database seems to be quite good because most of the time the database has only to respond to a finite set of predefined queries with some parameters.

The two databases handles JSON files and can export their data as JSON. Thus it is possible to convert a MONGOCLDB database to a COUCHDB at little cost. But since their semantics is a bit different one needs to write a small script in order to convert one format to another.

Both systems seem robust but if one needs to do a lot of processing on the database the map/reduce aspect of COUCHDB seems very interesting. Otherwise MONGOCLDB seems a good choice.

---

1. According to this ranking: https://db-engines.com/en/ranking/document+store
2. According to this trend chart: https://db-engines.com/en/ranking_trend/document+store