6 MongoDB

CS 425
Web Applications Development

Alex Petrovici

Content

Introduction to MongoDB / Introduction

Why make a monolitic database ?

Reading from several sources requires JOINs to read.
The combination is slow.

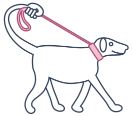
High computational cost.
Complexity grows exponentially with data volume.

A single data source containing everything would avoid combining.
High redundancy (greater volume) but it benefits from paralelization.

Introduction to MongoDB / Introduction

Document oriented

- Each element (document) contains everything needed to define the individual (with all its historical value)
- e.g., orders that include each price, which may vary
- Monolithic structure
- Focus: no form, free form



- Self-contained
- Very redundant

Introduction to MongoDB / Introduction

Collections

- A collection may contain anything.
- And we can have as many collections as we want.
- And several databases in the same server (centrally managed) but a distributed database would be better.
- Like other NoSQL databases, it is oriented toward unstructured storage:
  - each occurrence (individual) is stored in a BSON document
  - groups of objects are collections of documents (corresponding to the concept of relation or table)
- Being unstructured, it has no predefined design:
  - The collection is not defined: its documents can have any set of fields (even the same field name with different data types)
  - Database design is restrictive. In MongoDB, the global schema is not defined: the collection is created when the first element is added.

Introduction to MongoDB / Introduction

Collections

- Document: a set of data referring to an individual
- MongoDB uses the BSON (Binary JSON) document representation for data storage and transfer, which is a version of JSON that is more efficient in certain aspects.
- JSON (JavaScript Object Notation): a lightweight format for data interchange (it is a subset of JavaScript notation).
  - Advantage: simplicity (and efficiency with large volumes of data)
  - Disadvantage: not very secure
- To process both, the JavaScript (JS) programming language is used

Introduction to MongoDB / Introduction

BSON

- BSON (Binary JSON) `http://bsonspec.org/spec.html`, is a document exchange format based on the serialization (marking and encoding) of JSON documents.
- JSON is a human-readable exchange format stored in plain text files.
- A BSON file can contain one or more JSON documents (serialized in binary), which are structured as follows:
  - JSON documents are separated by commas.
  - Each JSON document is an ordered list of elements enclosed in braces … and separated by commas.
  - Each element is a triplet <name, data_type, value>.

Introduction to MongoDB / Introduction

JSON

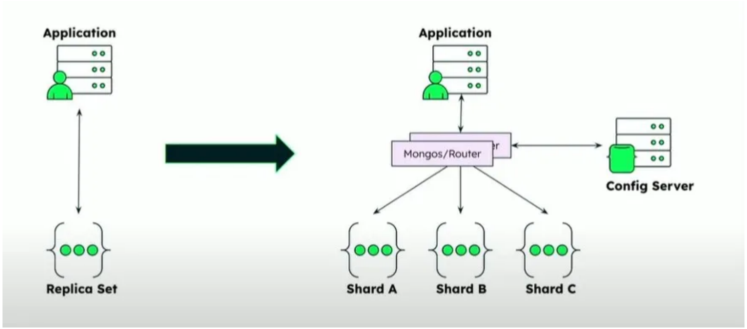
- JSON element: a triplet containing <name, datatype, value>
- name: a string (enclosed in double quotes, or not) that identifies the element (along with its data type).
  - The label can be repeated with different types: ”Age”:42, ”Age”:”young”
  - The name+datatype pair can be repeated in different scopes (subdocuments).
- datatype: is implicit (linked to the value, not the variable).

Introduction to MongoDB / Introduction

JSON

- Supported JSON data types:
  - Numeric: integers (32 or 64 bits); floating point (IEEE 754);
  - Alphanumeric: strings (enclosed in double quotes);
  - Explicit: null value (null); booleans (true, false);
  - Composite: subdocument (…); array ([array]);
  - Executable (parseable): regular expression; JavaScript code.
- Data types added in BSON: date; byte string.
- value: defines both the data type and the data itself.

Introduction to MongoDB / Introduction



Source:
https://ravisrc.medium.com/mongodb-say-no-to-sql-8ef7b94a6ae9

Introduction to MongoDB / Introduction

MongoDB Server

Instructions to install: `https://www.mongodb.com/try/download/community`
In Ubuntu/Debian you can simply do:

```
wget <check-the-updated-url-from-mongodb>
sudo apt install ./mongodb-org-server_8.2.2_amd64.deb
```

In Ubuntu/Debian is better to install deb packages with `apt` instead of `dpkg -i` because apt will register the packages in the system and will be easier to update/remove and handle dependencies.

Introduction to MongoDB / Introduction

MongoDB Client: Shell

Instructions to install: https://www.mongodb.com/try/download/shell
Ubuntu/Debian:

```
wget <check-the-updated-url-from-mongodb>
sudo apt install ./mongodb-mongosh_2.5.10_amd64.deb
```

MongoDB Client: Compass (GUI)

Instructions to install: https://www.mongodb.com/try/download/compass
Ubuntu/Debian:

```
wget <check-the-updated-url-from-mongodb>
sudo apt install ./mongodb-compass_1.48.2_amd64.deb
```

Introduction to MongoDB / Service management

[● ◀] **systemd**

Check Mongod status

Currently Services in Ubuntu/Debian are managed by `systemd`. The `systemd` is the init system (the process PID 1) and the `systemctl` is the CLI frontend.
Systemd was initially developed by Red Hat to replace Linux's conventional System V init in 2010.

- In 2011, Fedora Linux became the first major Linux distribution to enable systemd by default, replacing Upstart.
- In 2012, Arch Linux made systemd the default, switching from SysVinit.
- In 2015, Debian (and so Ubuntu) made systemd as default, switching from SysVinit.

Documentation: `https://systemd.io/`

Introduction to MongoDB / Service management

systemd

WORKING WITH SERVICES

`systemctl stop` *`service`*	Stop a running service
`systemctl start` *`service`*	Start a service
`systemctl restart` *`service`*	Restart a running service
`systemctl reload` *`service`*	Reload all config files in service
`systemctl daemon-reload`	Must run to reload changed unit files
`systemctl status` *`service`*	See if service is running/enabled
`systemctl --failed`	Shows services that failed to run
`systemctl reset-failed`	Reset units from failed state
`systemctl enable` *`service`*	Enable a service to start on boot
`systemctl disable` *`service`*	Disable service (won't start at boot)
`systemctl show` *`service`*	Show properties of a service or unit
`systemctl edit` *`service`*	Create snippet to drop in unit file
`systemctl edit --full` *`service`*	Edit entire unit file for service
`systemctl -H` *`host`* `status network`	Run any systemctl command remotely

Introduction to MongoDB / Service management

[● ◀] **systemd**

CHANGING SYSTEM STATES

`systemctl reboot`	Reboot the system (`reboot.target`)
`systemctl poweroff`	Power off the system (`poweroff.target`)
`systemctl emergency`	Put in emergency mode (`emergency.target`)
`systemctl default`	Back to default target (`multi-user.target`)

VIEWING LOG MESSAGES

`journalctl`	Show all collected log messages
`journalctl -u network.service`	See network service messages
`journalctl -f`	Follow messages as they appear
`journalctl -k`	Show only kernel messages

Introduction to MongoDB / Service management

VIEWING systemd INFORMATION

`systemctl list-dependencies`	Show a unit's dependencies
`systemctl list-sockets`	List sockets and what activates
`systemctl list-jobs`	View active systemd jobs
`systemctl list-unit-files`	See unit files and their states
`systemctl list-units`	Show if units are loaded/active
`systemctl get-default`	List default target (run level)

Sources:

- https://access.redhat.com/sites/default/files/attachments/12052018_systemd_6.pdf
- https://www.redhat.com/en/blog/systemd-commands
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd

Introduction to MongoDB / Service management

mongoDB

In a nutshell

Start the service

```
❱ sudo systemctl start mongod
```

Check the status of the service

```
❱ sudo systemctl status mongod
```

Stop the service

```
❱ sudo systemctl status mongod
```

Disable the service (so it won't start on boot)

```
❱ sudo systemctl disable mongod
```

Introduction to MongoDB / Service management

System recovery

- If the service has been interrupted abruptly, the next attempt to start the listener will produce the error `Unclean shutdown detected.` (This will be shown in the interactive console and in the event log file, if it exists.)

Introduction to MongoDB / Service management

To restore a valid state, there are two alternatives:

1. **Run `mongod` in repair mode to create a copy of the damaged database:**
   This procedure creates a new valid database in the specified directory, with whatever data it can recover from the original files (without destroying them):

   ```
   mongod --repair --repairpath <new_data_subdir>
   ```

   To start the listener, use:

   ```
   mongod --dbpath <new_data_subdir>
   ```

2. **Start `mongod` in repair mode to truncate the damaged database:**
   Similar to the previous method, but the recovered data will be written in the same subdirectory, replacing the originals (which will be destroyed). **Only use this method if you cannot recover data by any other means.**
   1. Manually delete the `mongod.lock` file from the data subdirectory;
   2. Run `mongod` in repair mode: `mongod --repair`

Introduction to MongoDB / Service management

Robustness in MongoDB

- To prevent file corruption issues caused by system crashes, MongoDB provides two mechanisms
  - **Journaling:** When this option is enabled, write operations in MongoDB are first stored in a log and then executed. This results in a slight loss of write efficiency, but increases robustness. The option is enabled using the `--journal` parameter when starting `mongod`.
    - **Note:** Journaling logs are stored in `<dbpath>/journal`, and can grow to the point of slowing down the startup of the listener. To re-initialize it, you can make a copy of that subdirectory immediately after creating it, and restore it (before starting the listener).
  - **Replication:** The database is simultaneously stored on several machines, providing robustness and efficiency (for educational purposes, replication can be set up on the same machine using different locations, but in production, use different machines).
    - When a member of a replica set (one of the database copies) becomes unstable, rather than repairing it, it should be synchronized with the other members of the set (using the `sync` instruction), thus restoring stability.
    - Queries can be executed in parallel, increasing efficiency.

Introduction to MongoDB / Service management

Limits

- The limits of a tool usually depend on the version. Currently:
- Database names must be non-empty strings up to 64 characters long and cannot contain the characters / \, ., ' ', ", $, *. On Windows, the extra chars <, >, :, |, ? are not allowed.
- Collection names must be non-empty strings that do not start with `system.` nor contain the $ symbol. It is recommended that the first character be a letter or the underscore _ (otherwise, you must use `db.GetCollection(...)`). The string `db_name.collection_name` cannot exceed 120 bytes.
- The namespace file has a default size of 16MB (can be increased up to 2048MB), which allows for more than 24,000 objects (collections and indexes).

Introduction to MongoDB / Service management

mongoDB.

Limits

- Field names should not contain the characters: `$`, `.`, or the null character.
- Documents can be up to 16MB in size (in BSON format).
- Nesting of subdocuments can be up to 100 levels deep.
- The number of documents is unlimited (except in capped collections, which may be up to 2^{32}).
- A database can have up to 16,000 datafiles of up to 2GB each (0.5GB if `smallfiles` is used).

Introduction to MongoDB / Starting with Mongodb

- Start the shell console in a command window by executing `mongosh`.
- By default, it will connect to the local machine (`localhost`), on port `27017`, and to the database located in the `/data/db` subdirectory. If you want to connect to another database, specify the parameters `--host`, `--port`, and `--dbpath` as needed.
- By default, in an empty data subdirectory, MongoDB creates three databases: `admin`, `local`, and `test`. The connection will be to the last one (`test`).
- You can check this by executing the following commands from the console:
  - `db` : shows the active database (probably `test`)
  - `show dbs` : shows the available databases (all except the empty `test` database)
  - `use local` : switches to `local` (check with the db command)
- To create a new database, simply run `use <newDB>`.
- A new database only appears in the list after data has been added to it.

Introduction to MongoDB / Starting with Mongodb

Collections

- As previously introduced, the data referring to an individual (an object) will be specified using a BSON document, whose maximum size is 16MB.
- `show collections`: command that displays the available collections.
- The collection is created implicitly when the first document is inserted into it.
- To insert a document into a collection, use the following instruction:

```
db.<collection>.insert(document)
// e.g., db.customers.insert(doc)
```

- You can also insert several documents in a single instruction (Bulk insert) by describing them as an array of documents. This procedure is more efficient than running the corresponding sequence of insert instructions:

```
db.<collection>.insert([document1, document2, ... documentN])
```

Introduction to MongoDB / Starting with Mongodb

Collections

- There is a method on the database to create empty collections (in the active DB): `db.createCollection()`
- It is usually used to create special collections, with a different nature than normal ones, such as capped collections:
  `db.createCollection("audit", \{capped: true, size:1000\})`
- To reference a collection, remember it is part of a database (the active DB). Example: to refer to the collection `ebooks`, you need to enter `db.ebooks`
- You can also reference a 'sibling' database, located in the same data directory:
  `db.getSiblingDB('tiny').ebooks`

Introduction to MongoDB / Starting with Mongodb

Collections

- Remember that collections are objects and have methods and functions that define their behavior. Besides the one we have already seen (`insert`), there are other methods to query their documents:
  - `db.<collection>.count()`: shows how many documents are in the collection
  - `db.<collection>.findOne()`: displays one document from the collection
  - `db.<collection>.find()`: returns a cursor with the documents in the collection
- The result of `find()` is another object (of type cursor) that supports additional methods:
  - `...find().sort(\{<field>:1\})` sorts the output by the specified field (1: asc, -1: desc)
  - `...find().skip(15)` skips the first 15 documents
  - `...find().limit(10)` limits the result to 10 documents
- Examples: type in the console `db.ebooks.count()` and `db.ebooks.find()`

Introduction to MongoDB / Data import and export

mongoimport

- MongoDB also supports importing data from files (JSON, CSV, and TSV).
- Data import is handled using a separate command-line tool: `mongoimport <options> <connection-string> <file>`

`--host <host>`	*Host (by default, localhost:27017)*
`--port <port>`	
`--db <database>`	*Database: required, use --db or URI*
`--collection <col_name>`	
`--type {json|csv|tsv}`	*file format*
`--mode upsert`	*update existing documents*
`--drop`	*drop collection before import*
`--file <source_file>`	*Don't forget the file!*

You can see more options at https://www.mongodb.com/docs/database-tools/mongoimport/

Introduction to MongoDB / Data import and export

mongodump

- To back up and restore databases, MongoDB provides the `mongodump` utility.
- `mongodump` allows you to create binary backups (BSON) of your databases, collections, or specific data, which is essential for safeguarding against data loss, migrating data to other systems, or creating development/testing datasets.
- Data dump is handled using: `mongodump <options> <connection-string>`

`--host <host>`	*Host (by default, localhost:27017)*
`--port <port>`	
`--db <database>`	*Specify database to dump (all databases if omitted)*
`--collection <col_name>`	*Dump only this collection (omit to dump everything).*
`--out <destination_dir>`	*Directory to save the backup*

You can see more options at `https://www.mongodb.com/docs/database-tools/mongodump/`

Introduction to MongoDB / Data import and export

mongorestore

- `mongorestore` restores data from BSON backups made by `mongodump`.
- Data import is handled using: `mongorestore <options> <connection-string> <directory or file to restore>`

`--drop`	*Drop the target collection before restoring*
`--host <host>`	*Host (by default, localhost:27017)*
`--port <port>`	*Port (by default, 27017)*
`--nsInclude=<namespace pattern>`	*Specifies a namespace pattern to restore (controls db and co*
`--dir <directory>`	*Directory to restore from*

You can see more options at `https://www.mongodb.com/docs/database-tools/mongorestore/`

Introduction to MongoDB / Cursors

Cursors

- If we now run `db.ebooks.find()`, it should return 102 documents (the two added earlier and the 100 random ones).
- The result of a query in MongoDB is called a *cursor*, which is an object with its own existence and methods. (You already know some cursor methods: `sort`, `skip`, and `limit`.)
- However, when processing the cursor, the MongoDB shell only displays the first 20 results by default (to avoid showing endless listings).
- If you call a cursor method, it will execute on the current cursor (the last one processed). In particular, invoking the `it` method will show the next 20 results (and so on, until the cursor is exhausted).

Introduction to MongoDB / Cursors

Cursors

- To conveniently handle a cursor, it can be assigned to a variable:

```
var myCursor = db.ebooks.find();
myCursor; // calling it like this will execute and display the first 20 documents
```

- The cursor object offers various methods, including sorting, limiting, and functions for iterating over results.
  - `cursor.hasNext()` — Returns `true` if there are more documents to display.
  - `cursor.next()` — Returns the next document in the cursor.
  - `cursor.objsLeftInBatch()` — Shows remaining documents in the batch.
  - `cursor.toArray()` — Returns an array with the documents left in the cursor (if used when initially creating the cursor, returns all results of the query).
  - `cursor.forEach()` — Applies a function to each element of the cursor.
  - `cursor.map()` — Similar to `forEach()`, but returns an array of results.
  - `cursor.pretty()` — Pretty-prints (indents) the JSON documents.

Glossary / Terms I

Glossary / Abbreviations I