In simulating results with many different model versions, parameters, etc., it is challenging to keep track of results. When my previous solution of a hierarchical folder structure seemed too limited, I looked into databases. Here's what I learned and did:

- I first looked into sql databases and particularly sqllite, which has a default python interface.
- Very briefly, I found sql to be too inflexible. The issues is that you have to really define your data structure beforehand and if you want to change it at a later point, this causes a lot of effort. This is in my experience not suitable for science where we will think of new aspects of our problems all the time and want some flexibility.
- Luckily NoSQL databases offer precisely that flexibility. I decided to use a document-based nosql database and chose mongodb, with a relatively large community (and thus a possibility to find errors/solutions on the web) and a python interface (pymongo) being important criteria.
- A key feature I wanted was a possibility to easily inspect plotted simulation outputs and tag the simulation by keywords specifying whether the simulation showed certain effects of interest. In an early attempt using sqllite, I could either store the image as a binary (BLOB) in the data base or store it separately and have an URL/local file path to it stored in the data base. Several GUIs for sqllite would then allow me to either preview the stored binary image or open it externally by clicking the stored link, so that I could inspect it and then manually edit my tag fields. Unfortunately, a GUI with this functionality was not easily found for mongodb. What I considered:
    - Humongous is supposed to have that capacity but is paid
    - NoSQLBooster for MongoDB can do it, but works badly in general, making my intended workflow slow and painful. A key issue for me is that it was for some reason very slow to retrieve any entries – I tried solving this by indexing the data base but felt that this was not the problem or at least not a viable solution.
    - Mongo-express is a web interface that supposedly has the functionality – needs to be set up, but then works from your local browser. It is lightning-fast compare do nosql booster, can preview images stored as binary data and you can edit entries, though the latter is a bit cumbersome. One might even try a solution of combining mongo-express for viewing and another GUI (e.g. mongodb compass) for editing.

Note: the deployment of MongoDB and mongo-express was done on linux. MongoDB could also be set up on Windows. Mongo-express probably could not, but can be interfaced from Windows when running on a linux server (see below). NoSQLBooster can also run on windows and interface a data base on a remote (linux or windows) server.

CAVE: Many of the solutions used below do not consider security risks. We considered this okay because we ran things on a server within a protected network that has no connection to the outside world. If you have different intentions, you will want to spend a whole lot more brain in security.

MongoDB

- Troubleshooting
    - The configuration file is in /etc/mongod.conf
        - It may be useful to change the dbpath to something on /data/ , since this has more storage and is an ssd. Note that this may cause access permission conflicts, though (see below).
    - if you are starting mongodb with systemctl start mongod (see installation instructions), you can check status with systemctl status mongod

        - a common error is exit flag 14 – this likely means there is an issue with the linux permissions to the files mongodb needs to write to. Mongodb creates a dedicated linux user mongodb upon installation that runs all its processes. Sometimes, mongod needs to access files that this user does not have ownerhip of, especially if you changed the dbpath in conf. The fix is to identify the relevant files and change their ownership to the mongodb user. The commands are:
            - for files: sudo chown mongodb:mongodb <filename>
            - for directories: sudo chown -R mongodb:mongodb <dirname>
            - where chown is linux's change owner.

          Typical candidates for wrong permissions are:
            - your custom dbpath and everything in it
                - within there, the WiredTiger files, also in the protected folder journals
            - /tmp/mongodb-27017.sock
            - Other than that, check the log (see below)
            - Note: For me, what appeared to happen is that mongod ran fine at first start but then while running, it created new files that were under root instead of mongodb ownership, so after stoping and trying to restart, these caused permission issues. This happened in a cascade of at least 2 levels.
    - more detailed status output is in the log file – by default in /var/log/mongodb/mongod.log (or you may have specified differently in conf).

Replication

Reduce OpLog size (default is 5% of disk space). OpLog stores commands and is used for re-syncing after a replica was down. The larger the opLog, the longer we can sync without manual interference.

Indexing

Indexes enable faster queries ('find') and sorting in your DB. You will likely want to use indexing at some point. This won't matter as long as your DB is small, but for me, a multi-field lookup (collection scan) in a 30k-entry collection takes ~5-10 sec. Indexing can get quite involved and it may initially seem like you need advanced tools such as compound and text indexing. For my DB, I was able to achieve significant speedups in multi-field requests just by having a single-field index on a field with relatively low duplicity – mongDB will then quickly filter the DB by this field and only execute the rest of the query on the subset that matches the indexed field. For the above 5-10 sec. query, I could thus reduce times to ~30-150 millisec. I used an ascending index on a hash in hex-format stored as a string for this, so apparently ascending/descending indeces on strings work and don't require text indexing (I suspect text indexing is when you want adanced features like case-insensitivity).

There's a useful intro on indexing (and performance issues in general):

https://learn.mongodb.com/learn/course/m201-mongodb-performance/lesson-2-mongodb-indexes/

Schema design considerations (read early-on!):

The schema is the way entries are arranged in your documents. In MongoDB, it is relatively flexible, but it is still a good idea to consider some things early-on during your DB creation.

- Field order: Mongo-express (see below) currently offers no good option to customize the order of fields in the view, but just shows them in the order that the first entry shown was

created (I think). So, if you need to visually inspect or manipulate certain fields but not others, it is a good idea to have those fields first when you insert entries to the DB, to avoid scrolling, later on. As long as you have less than 1000 (2000?), the free version of Studio3T offers relatively simple options to rearrange field order ("reschema")

- Compound Indexing. If you need compound indexing, you need to consider some things in your schema, especially when you're working with array fields (dicts within dicts in python) – see "Limitations" in https://www.mongodb.com/docs/manual/core/index-multikey/ - essentially, try not to have more than one array field that may be relevant in a find operation.
  Possible workarounds: 1) instead of adding array fields to index, enter the individual subfields, i.e. arrayName.fieldName. I have not tested this. I suspect it may require correct ordering of subfields. 2) – see hashing hyperparameters
- Text indexing: MongoDB only allows one index per collection to index text entries. This may e.g. prompt you to put certain entries you might otherwise distinguish by a text field into separate collections.
- Hashing hyperparameter settings. In manually exploring the DB, I will often identify a particular set of hyperParameters while browsing one type of simulation (e.g. a certain experiment) and want to see more simulations of this setting. Entering all prior settings into a query is cumbersome. My solution is that I hash the relevant hyperParameters and enter the hash into the db. Thus, to query entries of the same hyperParameters, I just have to copy-and-paste the hash. I may also use this to get around my issue that my already-existing DB has different sets of hyperParameters in multiple array fields and thus cannot easily use them in an index (see earlier point – untested at this time). Note: I did this manually. MongoDB provides a hashed index option – not sure if this can be used to the same ends.

Mongo-express

- One "easy" environment to install and run mongo-express is nodejs and npm, the latter being a package manager for nodejs (see mongo-express github page)
  - To check whether it is installed, nodejs -v ; npm -v
  - If not installed, you may find various instructions on the web.
    - Just installing npm with apt-get without nodesource did install an npm version for me, but this was apparently too old for mongo-express.
    - Installing newer versions often relies on nodesource
      - As of now, threadripper is Ubuntu 18.04 and nodesource is incompatible with that → no way forward.
    - There exist other instructions using homebrew/linuxbrew, but these sound outdated, as well.
    - What worked for me is to install nodejs via conda (conda install nodejs). It comes with a recent npm version, even though it doesn't say so.
  - I have not tried going via the other environments specified on mongo-express github page, since they seemed more advanced (e.g. require mongodb docker employment).
  - Note: mongo-express prompted some security issues upon installation. Since threadripper is on a protected network, we chose to ignore those – in an open network, you may choose differently.
- after installing, start mongo-express (by typing It in the terminal) – it will connect to an active mongodb database (if you used the defaults for your database, it will find it automatically – if not you need to specify custom ports, etc). It remains running and outputs an address to

access it, typically localhost:8081 – you can access the mongo-express interface by typing this in a browser (on the computer you're running mongo-express)

- due to the conda installation of nodejs, the user directory of mongo-express for me was not the default but hidden in /home/<username>/miniconda3/lib/node_modules/mongo-express – there, you can e.g. find the default.config and create a config.js (cf. mongo-express configuration instructions).
- When running mongo-express in default mode, you will only see a test database. This is because the default user has restricted access. What worked to get around this is that I configured an admin user for my mongodb instance (e.g. see her- this uses mongoshell, which you can access by just typing mongosh on the server when a mongodb instance is running: https://dba.stackexchange.com/questions/111727/mongo-create-a-user-as-admin-for-any-database-raise-an-error), then in mongo-express's config.js, set admin to true and configure basicauth user and password to that user's credentials. *CAVE*: basicauth is insecure. At the very least, you will want to create your admin with a password that can be leaked (to change password, see e.g. https://www.prisma.io/dataguide/mongodb/configuring-mongodb-user-accounts-and-authentication#how-do-you-change-the-password-for-a-mongodb-user) , since you have to store the credentials in plain text in mongo-express's config. Better would be to go via more advanced authentication, e.g. https://www.helpnetsecurity.com/2019/04/26/securing-mongo-express-web-administrative-interfaces/

SSH tunnel

- it may be convenient to access mongo-express from another computer. An easy way to do this is to establish an ssh tunnel. I.e. from your local computer's terminal (or bash on windows), type:
- ssh -YNL localhost:8081:localhost:8081 <username>@137.248.41.241
- -YNL is the adjunct for establishing a tunnel. The two localhosts are from and to which port (so needs to match the one mongo-express is running on, remotely and an open port on your local computer). 137.248.41.241 is threadripper's ip-address (can use with other host names) and for <username>, put in your user name. It will prompt you for your credentials and then just run without returning anything – you should then be able to access mongo-express by opening localhost:8081 in your local browser. If you have already configured ssh access to threadripper, instead of your threadripper credentials, it may ask for your ssh credentials. You may also explicitly specify the id-file by the typical adjunct -i <locationofkey>, where the default is -i ~/.ssh/id_rsa