Mongo: a NoSQL database

cc BY

Why a database?

We can slip a little data into a cookie ... but not much

Our user might return on another browser, without our cookie

Cookies are good for a session, but we need

- Something that lasts longer
- Something that can hold more data

and often

- Something to hold data shared by more than one user

BY

Relational database

Model data as a set of tables

relation in the mathematical sense: A set of tuples

+ Clean, well-understood semantics, not tied to programming language

- Extra work to translate from application object model to database model and back (SQL queries are I/O)

+ Transactions give clean, reliable semantics to concurrency

- Transactions may limit performance

CC BY

Transactions: Reasoning about Concurrency

T0:

x = 0

T1:

x = x + 1

T2:

x = x + 1

What are the possible outcomes if T1 and T2 are executing concurrently?

MongoDB: A NoSQL database

Modeled on documents, not tables

Each ‘database’ in Mongo is like a relation (table) in SQL

The elements of a database are BSON structures (like JSON, but binary)

Very limited concurrency control: Atomic writes, ‘eventual consistency’

(basically it’s broken, but we live with it because ...)

Scales very well to many servers running in parallel

CC BY

Accessing a Mongo database from Python

```
from pymongo import MongoClient

geojson = { "type": "FeatureCollection",
          "features": featurelist
          }

client = MongoClient(MONGO_URL)
db = client.tracks
collection = db.samples


...
request = { "id": feed }
record = collection.find_one(request)
if (record == None):
      record = { "id": feed,
               "last_query_time": nowstring,
                "messages": [ ]
                    }
      collection.insert(record)
```

Search key

Search for matching records

Create a record (in a JSON-like format)

Add record to database

CC BY

Accessing a Mongo database from Python

```
from pymongo import MongoClient

geojson = { "type": "FeatureCollection",
          "features": featurelist
          }

client = MongoClient(MONGO_URL)
db = client.tracks
collection = db.samples

...
request = { "id": feed }
record = collection.find_one(request)

collection.update_one( {"id": feed },
      {"$set": { "messages": messages,
                 "last_query_time": nowstring }})
```

Search key

Search for matching records

Modify a found record

CC BY

Configuring the URL

CONFIG.py:

```
#  Host and port of MongoDB database
MONGO_URL = "mongodb://tracker:enroute@localhost:9999/tracks”
```

Password for database user (not very secure!)

The database we want to access

Mongo has its own application-level protocol (over TCP/IP)

User in MongoDB users database (not *you*; a pseudo user)

Randomly chosen port assigned when you install MongoDB on ix

BY

'Installing' Mongo 2.4 on ix
(only if you must)

Current version is 3.0; there are differences, so you may need to write code that works in 3.0 on your local machine and 2.4 on ix

'mongoctl' is a local per-user installation script

To start on ix:

```
$mkdir etc  # If you don't already have one
$cd etc
$ mongoctl install
```

(you will be prompted for a new password, which should NOT be your password to ix)

Look in etc/mongodb.conf file for configuration information

More information in
https://www.cs.uoregon.edu/Classes/16W/cis399se/howto/mongo.php

Mongodb configuration

Your ~/etc/mongodb.conf file will look like:

```
dbpath=/home/faculty/michal/mongodb
port=9999
```

You start your database running with

```
$mongoctl start
```

and see something like …

```
about to fork child process, waiting until server is ready
for connections.
forked process: 17714
all output going to: /home/faculty/michal/mongodb/
mongodb.log
child process started successfully, parent exiting
Started mongod on port 9999
```

cc BY

Installing on Pi and development machine

For pi:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mongodb-server
```

then go have lunch while it runs

For your development machine:
See mongodb.com

Two programs:

- mongod (database engine)
- mongo (shell)

cc BY

You might also see (first time) ...

```
MongoDB shell version: 2.4.9
connecting to: 127.0.0.1:9999/test
switched to db admin
{
    "user" : "michal",
    "pwd" : "f6f736f36d2104da987a6316b5699db2",
    "roles" : [
        "userAdminAnyDatabase",
        "readWriteAnyDatabase",
        "dbAdminAnyDatabase"
    ],
    "_id" : ObjectId("5536abb232b7867b16f82278")
}
bye
```

This won't be what you typed. What is it?

cc BY

Connecting (manually) to MongoDB admin db

```
$ mongo --port 9999 -u michal -p xxxxxx admin
MongoDB shell version: 2.4.9
connecting to: 127.0.0.1:9999/admin
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
   http://docs.mongodb.org/
Questions? Try the support group
   http://groups.google.com/group/mongodb-user
> db.system.users.find()
{ "_id" : ObjectId("5536abb232b7867b16f82278"),
"user" : "michal", "pwd" :
"f6f736f36d2104da987a6316b5699db2", "roles" :
[  "userAdminAnyDatabase",  "readWriteAnyDatabase",
"dbAdminAnyDatabase" ] }
```

Adding a user (for your programs)

In version 2.4.9, we create users with the addUser method on a database (different from version 3.0):

```
> db.addUser( { user: "tracker", pwd: "enroute",
roles: [ "readWrite" ]})
{
    "user" : "tracker",
    "pwd" : "d339d2ed360bdec659ca232a0e095141",
    "roles" : [
        "readWrite"
    ],
    "_id" : ObjectId("5536aec06120e30bb54c3459")
}
```

Now we can quit the shell, leaving the server running in the background:

```
> quit()
```

Try it now ...

We will use a Mongo database in our final project