Victoria Coleman
Lyra Silverwolf

Software Systems HW3 Database Design and Implementation Report

1) Installation: It is free? What license is it under? Is it easy to install?

MongoDB is free and open source under Free Software Foundation's GNU AGPL v3.0 license. It was very easy to install on a Mac running OSX Mavericks using pip install and worked virtually out of the box. On Ubuntu, we tried installing it via the following:

```
$sudo apt-get install python-pymongo
```

While it may deceivingly look like the package that should be installed, it is not. Uninstalling that package and then installing using pip was effective.

2) Immersion: How hard is it to get started? Is there good documentation? Example code?

MongoDB is very easy to get started with, especially with the Python distribution PyMongo. PyMongo allowed us to use a language we were familiar with (Python) to easily communicate with our MongoDB. The API documentation for both MongoDB and PyMongo are fantastic and make it very easy to search for added functionality that you might desire. There are also some useful examples of more complicated topics on Stack Overflow, although Stack Overflow was less helpful for MongoDB than it has been for other types of coding.

3) Semantics: What kind of queries can the system handle efficiently? Is there a good match between these capabilities and the algorithms we want to run?

According to the MongoDB website, there are several reasons that MongoDB is a good choice for a database. First of all, it is a NoSQL type database, so instead of using tables it uses a structure of collections of documents. Each document is analogous to a SQL table entry, but they accept more types of data including other documents and arrays. The document structure also maps well to other programming structures.

MongoDB also advertises itself as "high performance" for having embedded reads and writes, indexes that can "include keys from embedded documents and arrays" and schema that make polymorphism easier; however, we did not directly observe these features, or, if we did, it was inadvertent.

Finally, MongoDB is easily scalable to very large applications spread across many machines. We did not observe this feature either since we were just using one database run locally on Lyra's mac.

4) Performance: How fast can we process basic queries? Will the performance scale up to large numbers of concurrent queries? Can we index the data or precompute partial results to speed things up?

We experienced some problems with processing speed in the beginning, but that was largely to us writing inefficient code. Originally, we tried to slowly fill the database by creating or updating entries as we parsed the .csv file, but that was incredibly slow, and we gave up after several hours of processing. When we extracted and organized the data in our process and then uploaded each entry once without having to query the database to see if an entry already exists, it was pretty quick.

Our python script took about 15 minutes to write 1,900,000 actresses with accompanying movies to the database. When we go to read the database, it generally takes less than a minute to get a match if the actors are connected by two movies or fewer from the few pairs we tried. We did not try multithreading, but that would have improved the performance, especially if many things could read from the database at the same time. Precomputing partial results would definitely speed things up (for example, if there was an actor-to-actor database instead of an actor-to-movie and movie-to-actor ones, fewer queries to the database would need to be made, which is a major bottleneck in the code timewise).

5) ACIDity: What guarantees can the DBMS make regarding data consistency, integrity, and related issues?

We believe that the database is well isolated and as durable as the other data on our hard drives since this is where the data is being stored. The database is an isolated thread on the computer that can only be accessed through certain methods outlined in the API. Furthermore, the MongoDB site offers a full set of pages about the security of your new database, including information about authentication and encryption.

We did not test the atomicity of the system intentionally, but there were several cases in which we realized that what we were writing to the database was wrong and aborted the process mid-write using ctrl-C in the terminal. In these cases the database did not display any errors and the data did not appear to be corrupted in any ways by our abuse.

Finally, within our short period of use, the database was incredibly consistent. When we ran the same code on it multiple times, it continued to display the correct actor, path or other result we intended for it to give us. It even was consistent across our two development environments.

6) C interface: If we implement the server code in C, we'll need a C API to the DBMS. How does this API look? Is it easy to get started? Does it seem to be mature and reliable?

To interface with the database in C, we used the code provided by Allen Downey in the git repo. The install instructions were relatively straightforward, although the tutorial.c code seemed hard to understand how to scale. In general, though the MongoDB C driver documentation seems very thorough and complete.

7) Code Implementation

In this part of the project, we got a basic Oracle working, and the code can be found in the following git repository:

https://github.com/vcoleman/ActorMovieWeb

If you already have a populated mongo database with an following contents

"ActorList": database (client)
    "movies": collection
        "movie" & "actors": pages
    "actors": collection
        "movies" and "actor": pages

You can just run the script script "recursiveSearch.py" in the top level directory.

It prompts the user to enter the start and end actor names (and prompts the user to re-enter the name if it is not a valid one as listed in the database), and prints the map from movies to actors that connects the two using a simple recursive search.

To run the code starting from a clean computer, the following steps are necessary (as seen in the README.txt)

To run this program there are several steps you must follow.

1. Go to ftp://ftp.fu-berlin.de/pub/misc/movies/database/ and download actors.list.gz and actresses.list.gz

2. Unzip these list files and place them into the inputdata folder of imdb-data-parser-master
3. Run imdbparser.py
4. Copy the files actors.list.tsv and actresses.list.tsv from the output data folder to the outside folder where the other scripts live.
5. Rename actors.list.tsv and actresses.list.tsv to actors.list.csv and actresses.list.csv
6. Run parselonglist.py with a MongoDB port open on your computer
7. Change parselonglist.py to take in the opposite script (either actors.list.csv or actresses.list.csv, whichever it has not already parsed)
8. Run parselonglist.py again.
9. Run parselonglist_movies.py
10. Change parselonglist_movies.py to take in the opposite script (either actors.list.csv or actresses.list.csv, whichever it has not already parsed)
11. Run parselonglist_movies.py again.
12. Finally, run recursiveSearch.py in the terminal to explore the oracle of bacon - the terminal will ask you to input two actors and it will give you the shortest path between the two actors.