Makin Bacon: Relational Database Implementation

Complete Project can be found at: https://github.com/thomasnat1/MakinBacon

Our process:

We started by piping the imbd.py script output into a text file. We then wrote node.js code with mongoose to read the text file line by line. We would check the database for each actor and movie and if an actor existed, it added the movie to their list and if the movie existed, we would add the actor to it. We ran this but because we were using node.js, it would attempt to run all the lines in at concurrence and access the database at the same time. This broke the implementation, which was when we decided to use SEMAPHORES! This allowed us to restrict concurrent access to the db and it worked as intended.

The issue with this implementation was that it ran really slowly because it had to check if either actor or movie already existed. Our estimate put the total computing time at around 300 hours. We decided to go with a new approach of creating a full actor with all their associated movies and then just adding the actor to the db. This was doable with the text file we had. We then needed to do the same for movies. This required us to edit the imdb.py script to output the movie and then the actor, which we then alphebetized, using http://textmechanic.com/Big-File-Tool-Sort-Lines.html. We could then do the same thing we did to the actors. Using this method, it took only several minutes to load in the entire database.

For the database query, we were able to do it easily in the node.js with the mongoose interface. For the C interface, we were able to modify the tutorial code and write a function that takes in an actor name, finds the actor in the database, and then prints the actor's name and the names of all the movies that actor is in. We are confident that we could do the same for movies with no trouble, and that we could also create an array of movies for each actor (or actors for each movie) and return that from our function rather than printing it. This would then allow us to execute a breadth first shortest path search algorithm using those two database queries, which would take a little bit of work, but would also presumably be roughly the same across databases, barring of course databases with built in shortest path search algorithms, which may be a better choice for this application.

Installation:

To get the database setup, you start with the the text files found here:
https://drive.google.com/file/d/0B9Iv5Ug-zleoSkZBVmF1T0ZZNjA/edit?usp=sharing
https://drive.google.com/file/d/0B9Iv5Ug-zleoVGMxTV9CREo5UEk/edit?usp=sharing

Our implementation relies on Node and MongoDB. To get setup with Node, check out http://nodejs.org. To get setup with MongoDB, go to http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/ and follow instructions.

For our database interface, we use Mongoose. For that and all other project dependencies, simply run:

npm install

Which will install all dependencies.

Once that is setup, grab the readDataIntoDB.js and, get the allActors.txt and allMovies.txt files from https://drive.google.com/file/d/0B9Iv5Ug-zleoNGU3Nkt6Qk83aFE/edit?usp=sharing and https://drive.google.com/file/d/0B9Iv5Ug-zleod0VId3NKdnJNbFk/edit?usp=sharing, then run:

node readDataIntoDB.js

For the c interface, make sure to get the mongo-c driver, which you can do by executing these instructions in a UNIX terminal:

```
git clone https://github.com/mongodb/mongo-c-driver.git
cd mongo-c-driver/
git checkout v0.8.1
make
```

Be sure to update the LIB in Makefile to point to the directory in which you installed the mongo-c driver

Immersion:

Using our initial approach of loading in the database from a single file, we estimate the loading would take approximately 300 hours. Luckily, we found an alternate approach by generating two different text files, one with the actors in alphabetical order and one with the movies in alphabetical order, which cut out the need for any database interaction other than saving, speeding up the data upload significantly.

This should start the process to setup the mongo database, which should take several minutes.

The code to make a query is currently setup to run a server that can be accessed using

localhost:300/actorName/(Any Actor)

To start this server, simply run

node server.js

The mongo-c driver code needed for queries is fairly complex by comparison to what you need in node, but there is fairly good documentation and example code, as well as some discussion of how to do things on stackoverflow for common queries.

Semantics:

The system can handle any query to the database for any Actor or Movie using mongoose syntax like:

```
Actor.findOne({name: actorName}).exec(function(err, actor){
    console.log(actor);
    console.log(actor.movies);
```

The structure is setup so that we can load an actor, find the movies they are in, then find each of those movies in the db, and access every actor for each movie. This is a good match because it will allow us to eventually set up a fairly efficient shortest path breadth first search algorithm that will require relatively database accesses.

A graph based DB might be better suited for our purposes as they tend to be better at shortest path searches, but we can definitely do what we need to do, even using the c interface rather than the simpler javascript implementation.

Performance:

Our implementation should be able to run fairly quickly. We will be able to query the database to get the whole list of either actors or movies that want at each node in the search tree. Additionally, we were able to do a Node.js implementation, meaning that we will have excellent concurrency capabilities of handling many queries at once.

Our c prototype only has very limited functionality, but it runs very quickly and could be expanded to fully implement the desired functionality, which should also run at acceptable speeds.

ACIDity:

MongoDB is a very reliable DBMS with a consistent history of reliability. Additionally, no code outside of the readDataIntoDB.js has any capability to edit the database. After running the readDataIntoDB.js script, we validated the data of a couple of records against the imdb website and found that they matched.