CIS 550 PROJECT REPORT - MAX SCHEIBER, BRIAN SHI, AND RIGEL SWAVELY

1. Technical Challenges

Throughout the course of the project, we encountered several technical challenges. The first of such challenges was deploying our web application to the cloud. Although all members of our team had worked on developing web applications in the past, none of us had used platforms similar to Amazon EC2. Figuring out how to set up the account properly with the correct security and firewall settings proved to be a bit of a challenge, but after carefully following the instructions provided through the course we were able to successfully log in to a server. Finally, once we set up the proper software on the server such as Node and Git, we were able to start the Node server and access our application.

In addition, connecting to Oracle from our web application proved to be fairly difficult. Although there are commonly used, well maintained Node packages to connect to most database systems like PostgreSQL, MongoDB, Redis and more, there is not much support for connecting to an Oracle database. Thankfully, we were able to find one package, but installing it turned out to be fairly difficult. The package required software that had to be separately downloaded from Oracle in order to work, and configuring it took several tries both on our local machines and on the server. Once it was finally set up, however, using the interface to interact with the database was relatively straightforward.

Finally, we had some trouble creating a schema for the database that suited our requirements. We wanted to have one common interface to go between our different media types, such as trips, destinations and photos, and user posted content regarding those media types, such as ratings and comments. Additionally, we wanted to avoid having multiple likes, commments, and ratings tables for each type of media (eg. photo likes, trip likes, etc), and instead have one common set of tables among all of them. However, that would require a relationship from every media table to the ratings, likes, and comments tables, which still seemed wasteful. Additionally, if we were to implement another type of user created content, such as hashtags, it would require changing every table to include a new relationship under this implementation. Instead, we decided to have one central Media table, which had references to every entry in each of our specific media tables. The media table was indexed by a primary key which was a composite of the relevant media's type and its pid in its own table. For example, if we had a photo with primary key 12 in the Photo table, its corresponding primary key in the Media table would be ('Photo', 12). The

Date: 12.16.2014.

ratings, likes, and comments tables then simply used the primary key of the entry in the Media table that it referred to, minimizing duplication.

2. Performance Evaluation

We used Apachebench to benchmark our application. We hypothesized that the main bottleneck of our application was the constant calls to Oracle - we assumed we were IO-bound on the server's end, rather than IO-bound at the client's end (i.e. network latency) or CPU-bound. We therefore conducted several tests to determine whether our hypothesis was correct.

2.1. **Local versus EC2.** We first attempted to control for running our application on `localhost` versus hosting it on Amazon. We therefore ran 100 sequential requests to load the profile of username `rigel`, who had a lot of photos and trips.

Description	Mean (sec.)	SD (sec.)
`localhost`	3593	705
EC2	408	199

From this, we saw an interesting effect of spatial locality. We would have expected the `localhost` calls to be quicker. However, the fact that the Oracle database and EC2 instance are both on us-west, combined with the simple nature of the EC2 server being beefy, made the cloud calls much quicker. This surprised us, but it makes sense in retrospect.

2.2. **CPU versus IO.** We next attempted to discern whether our latency was CPU-bound or IO-bound. We therefore ran 100 sequential requests to load the profile of username `max`, who had few photos and trips, and the profile of username `rigel`, who had a lot of photos and trips.

Description	Mean (sec.)	SD (sec.)
`localhost`, `max`	1017	102
EC2, `max`	283	60.6
`localhost`, `rigel`	3593	705
EC2, `rigel`	408	199

From this, we can infer that a lot of latency comes from the Oracle database. Loading all 42 of `rigel`'s photos added a quarter of a second to the page load time on average. This is also likely the reason that `rigel` had a higher deviation on page load times.

2.3. **Sequential versus concurrent.** *Note*: This section is for extra credit, as delineated by the writeup.

Let us explore the time needed to load `rigel`'s profile as a function of concurrent requests.

Concurrency	**Min (sec.)**	**Mean (sec.)**	**SD (sec.)**	**Median (sec.)**	**Max (sec.)**
1	311	408	199	338	1843
2	312	427	201	372	2075
3	316	500	233	429	2600
5	327	688	492	578	6925
10	351	744	313	672	3586
15	416	1154	626	968	6492
25	555	1773	806	1601	9173

Methodology: We made requests like `ab -n 1000 -c 10 [url]`, where we kept the number of total requests ten times the number of concurrent requests.

Our application can handle concurrency reasonably well; it handles up to 10 simultaneously heavy requests with reasonable latency. Node's event-driven nature facilitates this. We conjecture that Oracle is not able to handle more than ten simultaneously requests gracefully; we do not have hard evidence of this, but judging by our server's logs, Oracle started clumping together similar database calls at this point. For example, the logs would have seven or eight consecutive responses from the `Photo` table, even though we also made subsequent calls to the `Trip` table (among others).

We finally noticed the long tail on these statistics. Our application will every now and then take a preposterous amount of time to load a page. We aren't positive why this is, but speculate that it may be because of quirks in the OS scheduler on the server or database.