ECS 116 Assignment 3 Report

Nisha Thiagaraj, Connor Young, Cindy Chen

June 9th, 2024

1 Teammates

Nisha Thiagaraj, Connor Young, Cindy Chen

2 Statement about ChatGPT

I, Connor Young, use the GitHub Copilot extension in VS Code. I primarily use Copilot as a better auto-complete when writing my code. I find it most helpful when I want to explicitly write repetitive code instead of using a for loop when making lots of graphs for example. In this situation, Copilot provides me better results since there is more context for it to use when making suggestions. I do not rely on Copilot when writing specific functions because the suggestions are often not what I want or completely wrong. There are occasions when Copilot gives a suggestion that gives me new ideas for my code, but it is very rare that I do not change the code Copilot suggests. Overall, I've found that GitHub Copilot helps to speed up my coding, but it has not changed the way I code nor do I rely on it.

I, Nisha Thiagaraj, used ChatGPT occasionally to help with some of my errors when I got them while compiling my code for each notebook. ChatGPT helped point out errors and gave me ways to rewrite the code so that I did not get those errors anymore. Sometimes, ChatGPT was not very helpful and my errors would not get solved, so I would have to solve them myself or ask my group to troubleshoot. Overall, I was able to successfully run all my code after using ChatGPT as an occasion assistant. I did not use any other LLMs.

I, Cindy Chen, did not use any LLMs for my code.

3 Statement about distributed work

We primarily worked independently on Step 1 (Problem Set 5) through Step 4. We worked together to cross-check and reference our results, as well as to troubleshoot any issues we were having. For Step 5 (extra credit), Connor worked on the queries that we then each ran independently. For the report, we each answered two questions: Nisha answered Question 1 and 2, Cindy answered Question 3 and 4, and Connor answered Question 5 and 6.

4 Comments on Observed Performance

4.1 Question 1

Since the pipeline works on left joining of db.listings with db.reviews, this means that to run the pipeline, there will be a scan of reviews for each records that is in listings in order to match up each id in order to join the collections. There are 39202 records in listings which means that running the pipeline will mean that MongoDB scans reviews fully 39202 times.

To compute the time it takes for MongoDB to make one full scan of the db.reviews collection, we would divide the 39202 scans by total scan time without index of 4 hours to see how much time it would take for just one scan of reviews. This comes out to be about 0.367 seconds on average.

To compute the time it takes for MongoDB to perform an index-based retrieval of all documents in db.reviews having a particular listing id value, we would divide the 39202 scans by the total scan time with index of 2 minutes to see how much time it would take for an index-based retrieval of all documents in db.reviews having a particular listing id value. This comes out to be about 0.00306 seconds on average.

4.2 Question 2

For part 2 it did not seem that an index was necessary because of the structure and complexity of the calendar data versus the reviews data. The reviews data has the `comments` field which is harder to sort through and process since it is more unorganized and text-based. However, with the calendar data, it has more straightforward and numerical data which is easier and faster to process and filter through. Even though there are more entries in the calendar.csv, the content in each entry makes an index less necessary compared to the reviews.csv.

4.3 Question 3

From lecture, we learned that NoSQL systems such as MongoDB have high-performance data access, meaning that retrieval of documents is very fast despite the large number of documents. The NoSQL system is highly optimized for key lookup, which is related to sharding. Sharding is when the data is broken into subsets, and each shard is identified with a shard key which allows for queries to run in parallel across a large amount of data.

Additionally, we would like to note that there is actually an existing index for this part of the assignment, as we had previously created an index on the "listing_id" column of our `reviews` collection. The `reviews` collection was then used to created `listings_with_reviews_m`. We aggregate on this collection in Part 4.

4.4 Question 4

Indexing can improve run-time in this case, as our query was not range-based. Indexing can sometimes slow down range-based queries, as those are usually fast even without indexes. However, in our collections, "id" is a string that is not previously sorted, so adding an index should improve the run-time for Part 4.

4.5 Question 5

Creating the `result` cursor is so quick because the cursor is just a pointer to the result set of the query. Creating the cursor does not make a request to the database and is essentially a description of our query that we use to make requests to the database. In contrast, creating `list(result)` iterates the cursor, makes requests to the database, and fetches all of the document data. This process is much more intensive as requests are sent to the database and data is sent to the user, so it makes sense why this process takes so much longer than just creating the cursor.

4.6 Question 6

Note: The run times mentioned in this answer are based on Connor's machine.

From our informal results, we found that using the text index did improve performance. The time to create `list(results)` for Query 5 pos saw a 70.5% decrease in run time (34s to 10s). The time to create `list(results)` for Query 5 neg saw a 99.5% decrease in run time (569.5s to 2.9s).

We saw similar results when using tsv-based text indexing in Progrmming Assignment 2. The results of those experiments showed that using text indexing significantly improved performance and reduced run times.

5 References

We did not use any outside sources.