Stephen Brown

CS 453 Project Report

Team Members: Tyler Bounds and Chris Hight

Due 12/04/16

Introduction

During the time of this project, we've done several tasks over the last few months to build up to our final implementation of the project prototype. For the first part we did data modeling, where we designed and wrote our queries in different modeling formats: Key/value, document, and column. For the second part, we wrote about two different distributed data management systems and compiled a system profile about them. For the third part, we designed our data queries in one of the systems we profiled in part two. Unfortunately, neither of the systems we profiled we ended up using due to technical issues, which are further explained in this document.

For the fourth (and final) part of this project, our team decided to use MongoDB as our distributed data management system. This decision was made primarily because of MongoDB's ability to import .csv files, which is the format the data was in. In addition to the ease of importing .csv files, the syntax of MongoDB is fairly straight forward and isn't very difficult to get the hang of with some practice. MongoDB also uses document-based stores, which works out perfectly for us since we're able to store all of our documents in the form of JSON objects (or BSON, as MongoDB uses), which makes querying fairly straight forward and easy to understand.

After a couple hours spent getting everything started (MongoDB server and client running, and importing data), we were able to start writing our queries.

Contributions

For my contribution for this project, I spent a decent amount of time researching different distributed data management systems and writing up test queries. I've had a fascination with DDMS's ever since I interned at the Home Depot Quote Center and used Elasticsearch to build an analytics website for their customers. We used a .NET framework along with SQL and Elasticsearch's C# NEST client to help pull data out of the aggregated queries. Unfortunately, Elasticsearch didn't fit our needs for this particular project, since it's meant to query data out of a single index, and there is no way to do subqueries, or at least no efficient way. So I set out to find a solution to our problem.

Major Decisions

One option to make Elasticsearch work was to combine all 4 .csv files into a single .csv file and then index that into Elasticsearch for querying on. After some research, I found that this would be possible, but at this point would require more work and time than just finding another DDMS that would work for our needs. After all of this, we decided to head in a different direction from Elasticsearch and decided to go with MongoDB.

This match was perfect because MongoDB made it quite easy to import .csv files, and a lot of the syntax used for writing queries is in Javascript, which meant we didn't have to spend time learning a whole new language. In addition to this, we were able to insert into our database four different tables for the four .csv files: freeway_detectors, freeway_stations, highways, and of course the behemoth freeway_loopdata (with just shy of 18 million rows of data).

I originally spent many hours (close to 20) working with Elasticsearch's Logstash plugin in order to figure out how to import .csv files, and even then it took nearly two hours for it to fully index the freeway_loopdata. MongoDB was a breeze for this (after fixing some formatting issues with a couple of the files), and imported freeway_loopdata in about 20 minutes.

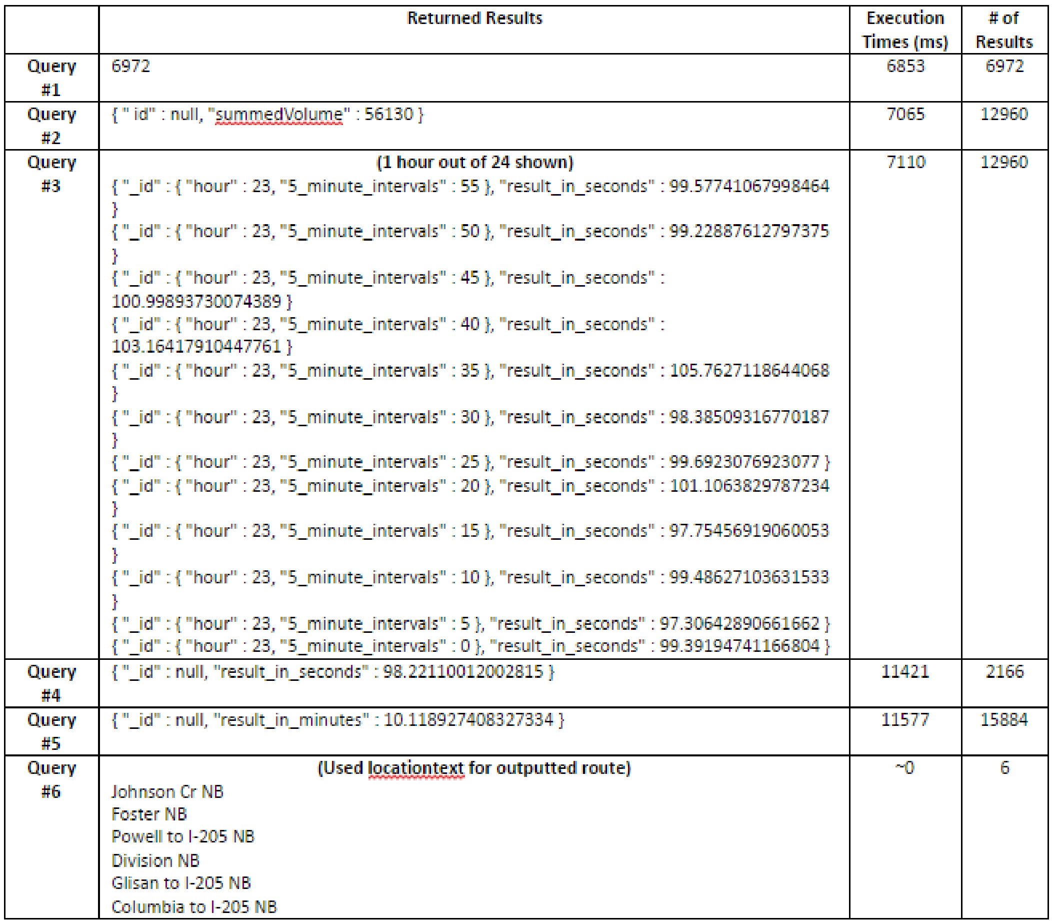
Just from seeing the ease of importing data, I had wished I hadn't spent so much time on trying to get Elasticsearch to work, and instead had went straight to MongoDB. But we made the correct choice in leaving Elasticsearch and going with MongoDB, and we're happy that it was able to work out so well.

Along with the ease of importing, one major advantage MongoDB has is its ability to store query results from collections into variables, and then to be able to use those variables to query from other collections. With this amazing ability, we were able to make use of the four separate .csv data collections without having to deal with any messy or time consuming joins or semi-joins.

After all this had been established and we had gotten the hang of the syntax, it was finally time to start writing the queries.

Results

For the results of this project, we decided to try to meet the stretch goal and to do all six of the queries running on the full dataset. So after quite some time and effort, we were able to meet this goal and implement all six of the queries using MongoDB. The total number of rows imported for all documents is 17,908,260, where the main contributor is freeway_loopdata. Below is a table of the summarized results:



	Returned Results	Execution Times (ms)	# of Results
Query #1	6972	6853	6972
Query #2	{ "_id" : null, "summedVolume" : 56130 }	7065	12960

Query #3	**(1 hour out of 24 shown)**
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 55 }, "result_in_seconds" : 99.57741067998464 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 50 }, "result_in_seconds" : 99.22887612797375 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 45 }, "result_in_seconds" : 100.99893730074389 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 40 }, "result_in_seconds" : 103.16417910447761 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 35 }, "result_in_seconds" : 105.7627118644068 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 30 }, "result_in_seconds" : 98.38509316770187 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 25 }, "result_in_seconds" : 99.6923076923077 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 20 }, "result_in_seconds" : 101.1063829787234 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 15 }, "result_in_seconds" : 97.75456919060053 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 10 }, "result_in_seconds" : 99.48627103631533 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 5 }, "result_in_seconds" : 97.30642890661662 }
	{ "_id" : { "hour" : 23, "5_minute_intervals" : 0 }, "result_in_seconds" : 99.39194741166804 }
		7110	12960

Query #4	{ "_id" : null, "result_in_seconds" : 98.22110012002815 }	11421	2166
Query #5	{ "_id" : null, "result_in_minutes" : 10.118927408327334 }	11577	15884
Query #6	**(Used locationtext for outputted route)** Johnson Cr NB Foster NB Powell to I-205 NB Division NB Glisan to I-205 NB Columbia to I-205 NB	~0	6

MongoDB has a good way of obtaining statistics from queries by using .explain("executionStats"); to display the execution statistics. From this you're able to see "executionSuccess", "nReturned", "executionTimeMillis", "totalDocsExamined", and more. It also breaks down queries when using "and" and "or" by showing you the results for each subquery.

When creating variables to hold query results, the amount of execution time is basically zero. This is because all of the data sets except freeway_loopdata are so small that MongoDB has no problem running queries on them. This is why the execution time for query #6 is approximately zero, since it didn't involve freeway_loopdata and only involved freeway_stations. All of the other queries involved freeway_loopdata, which is the main reason they have execution times greater than zero.

Even when you consider the maximum execution time of 11577ms for query #5, it's pretty incredible that it can actually query based off of the detectorid's and starttime parameters and out of nearly 18 million rows of data, it can pull out all of the results in mere seconds. This definitely shows how extremely fast and flexible DDMS's can be.

Out of all of the DDMS's that we could've chosen for this project, I'm glad we chose MongoDB. I know there are a lot of systems to choose from, and I don't regret choosing this one.

Lessons Learned

During this project I've learned quite a bit. I've learned about different data models, designing and writing queries, and how to implement and use a DDMS. I started initially with a DDMS (Elasticsearch) which unfortunately didn't work out, but was able to learn from it and move on with a different DDMS (MongoDB) that worked for my needs.

It's frustrating when you spend a lot of time working on a project, and all of that time spent leads nowhere before you find something that actually works. We also spent a lot of time figuring out ways to pull data out of queries and to be able to aggregate on that data. But when all is said and done, and you're able to successfully finish something you've spent so much time on, it feels great to finally finish and feel that you've achieved something.

One main thing I learned while writing our queries, is that we never did it perfectly the first time, and would often find ourselves going back to previous queries and fixing or changing certain parts of them to optimize them and make them more efficient or compact. After a while we had to stop going back to change them, and instead realized that there are so many different ways to design them, that we had to decide to leave them alone at some point, otherwise we could spend all our time optimizing instead of just finishing it.

This process happens a lot in Computer Science, but I do enjoy it, and even after all the time spent on the project for this class, I appreciate the knowledge I'm coming out with. After all the setting up is finished, and I'm able to just sit down and write queries, I realize how much joy I actually get out of it. It's like a puzzle you have to solve, and writing a query that returns proper results is a great feeling.