Query Performance Comparison of SQL vs NoSQL

Project Proposal

Bradley Culligan
cllbra005@myuct.ac.za
University of Cape Town
Cape Town, South Africa

David Court
crtdav015@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Relational databases are the current de facto standard for database implementations but they struggle to perform optimally when utilising big data. NoSQL databases can be categorised into various types which are each well-suited towards specific needs of big data. A suitable utilisation of relational databases and the various types of NoSQL databases can allow for better overall performance. We propose a project to compare the performance of MySQL, Cassandra, and MongoDB when performing filter and aggregation queries. This research can assist in designing systems that implement NoSQL with modest resources.

KEYWORDS

Database, SQL, NoSQL, Cassandra, MongoDB, MySQL, Column, Document, Performance

1 PROJECT DESCRIPTION

The big data era has brought challenges to the traditional relational database model. Big data was originally characterised by three traits [12]:

Volume The quantity of data.
Velocity The rate at which data is created.
Variety The structured, semi-structured and unstructured notion of data.

This definition has been extended to include more characteristics such as *veracity* and *variability* which relates to the truth of the data (possible errors) and the flow of the data respectively. For the purposes of this project, the problems in the relational model were made clear by the original three traits (volume, velocity and variety) so we shall limit our discussion to these characteristics.

Web 2.0, the Internet of Things (IoT) and digitization has led to vast quantities of data — too vast for the relational model to handle [10]. Relational databases have proved to not scale well to this volume of data. The velocity of data also requires that the relational model is able to perform fast transactions. However, the poor scalability and complex concurrency techniques have caused a bottleneck on performance. Further, the relational model requires strict schema which is not well-suited towards the variety of data in big data. The NoSQL model has been developed to better suit big data and directly appeals these issues.

The NoSQL model covers a broad range of database models, each with their own strengths and weaknesses. Since their inception, time has allowed them to mature and become solutions used in the modern data focused world. However, while NoSQL may offer solutions to certain problems, they are not able solve all the problems alone. Polyglot persistence has been deemed the solution, allowing various data models to be used such that the right tool can be used for the right job. This creates a database system that is able to handle a larger variety of data problems and better optimise those solutions by using a data model which will have better performance [11].

NoSQL (Not Only SQL) encompasses a large class of databases which typically do not conform to the relational model [6]. In general, we can divide the NoSQL class into four core data models but this project intends to focus on two of the available databases:

Document Store This model uses a key for lookup but the value is semantic (being semi-structured) and normally stored in JSON or XML format [8]. Examples include MongoDB and CouchDB.
Column Store This is most similar to the relational model, but data is stored by column rather than by row. Examples include Apache Cassandra and HBase.

Database implementations are often limited to relational databases because there is a lack of knowledge about alternative database choices, or specific NoSQL databases are implemented because they fit into a developer's technical stack (such as MERN/MEAN where the database used is MongoDB). Thus, poor awareness of the effects of database choice can lead to unoptimized database performance or queries which fail to run in a feasible amount of time. Performance analysis research can assist in making informed decisions when designing polyglot persistence architectures for individual projects. This is of particular importance as big data becomes more prevalent and our world becomes more data driven, thus, efficient database solutions are expected.

2 PROBLEM STATEMENT

Database architectures need to be designed on a per project basis. Thus, we focus the scope of this project on comparing the performance of MySQL, Cassandra, and MongoDB when performing filter and aggregation queries. Note that MySQL was chosen as the relational database (representing the de facto standard of database implementation) which sets a baseline in the comparison. Cassandra and MongoDB are the specific implementations we use to compare the NoSQL performance. These are the top ranked NoSQL databases within their respective categories[1].

Cassandra is designed to handle large volumes of data spread across many commodity servers while still providing a highly available service which has no single point of failure [13]. Cassandra is also well-suited towards operations which perform aggregation over a large number of fields, often for analysis. Modern datasets which exhibit these properties include data from IoT devices and

[1]db-engines.com/en/ranking

site-tracking data. These datasets typically have many columns to perform aggregation on, making them an ideal choice for understanding our comparisons.

MongoDB's flexible storage format makes it a natural fit for product catalogue systems that require capacity to store many different types of objects with different sets of attributes [2]. Documents in MongoDB demoralise the entity data which is ideal for product catalogue data because we can retrieve all required product data in one quick read. Further, fields can be indexed for faster filtering. Lastly, MongoDB is designed for high horizontal scalability using sharding, making it ideal for storing large quantities of products.

Research reviewed prior to this project revealed a few gaps. Firstly, the research which focused on studying the performance of types of NoSQL were mostly published when NoSQL was first becoming popular. However, NoSQL databases have matured in many respects and have increased performance in their modern implementations. This limits the research's use because the results may no longer hold valid. Further, the performance benchmarking was often done using multiple NoSQL databases without appropriately modelling the databases. Thus, the databases were used generically and fail to capture analysis on the effects that database modelling have on the query performance. Further, if the research did not focus on general performance benchmarking, it focused on specific use cases. There is currently a deficiency in research on the database and dataset combinations which we have outlined and intend to fill this gap.

Therefore, this project will generate updated research on performance benchmarking for MySQL, Cassandra, and MongoDB where the focus is on their query performance under the stipulated datasets. The datasets have been chosen to highlight the performance benefits when using the best suited database. That is, Cassandra is expected to be performant when performing queries on datasets intended for aggregation (based on the large number of columns) and MongoDB is expected to be performant when performing queries on datasets intended for filtering (based on the nested structure of the data). This allows us to compare the performance of these databases with the required query types — assisting in fulfilling the first gap recognised. The comparison of these results will be discussed in the context of their respective datasets and highlight the modelling choices appropriately required for the types of queries and dataset used. We emphasise here that the dataset only has read queries applied to it, simulating a stored dataset that is not being updated but only queried on for retrieval.

Our research intends to answer the following research question — with regards to a modest data size and resource allocation, we intend to measure:

- To what extent is latency and throughput performance of *Cassandra* better than MongoDB and MySQL for *aggregation* queries of *IoT* data?
- To what extent is latency and throughput performance of *MongoDB* better than Cassandra and MySQL for *filtering* queries of *product catalogue* data?

Notice that the research questions have assigned a particular database as performing better when assigned their associated dataset. This is in tandem with our research design to match optimal datasets and databases according to the literature recommendation. However, the minimal amount of research on NoSQL with modest resource allocation could imply that we shall obtain unexpected results. Therefore, we cannot stipulate the extent to which we require our databases to exhibit a performance advantage in order to deem our research a success, but rather any performance insight will be of value.

3 RELATED WORK

With the explosion in interest and implementations of NoSQL databases around 2010, many works were written to compare how each differs in performance, structure, and their respective strengths and weaknesses [2, 3, 7, 9, 14]. These papers are now relatively old and, therefore, require an updated analysis as the tools have matured. Performance benchmarks with various system resource allocations were analysed and this often led to performance impacts. Performing our benchmarks on multi-nodal systems will allow us to gain further depth in our analysis as this is when Cassandra's performance capabilities are better exhibited [4]. Research released at the time also outlined the design and architecture of the NoSQL systems coming into fruition, such as the creation of Cassandra [13]. Further, research was released on understanding the NoSQL features in general (such as BASE and the CAP theorem) which highlight some of the key differences between NoSQL and SQL systems [5]. Finally, we have reviewed research on polyglot persistence which explained the need for developing research in this area [11].

New benchmarking performance tools were required to be developed to effectively perform benchmarks on the new databases that were being created. Some of the new benchmarking tools created include YCSB [1], TPC (alternative implementations), and NoSQL Bench. YCSB has since become the de facto standard benchmarking tool and will be used in our research.

The research papers on performance comparisons assist in calibrating our research with what has historically been achieved. The papers also highlight best practices and allow us to avoid techniques or tools which have proven to be poor.

4 PROCEDURES AND METHODS

We discuss an outline of how the project will proceed, the benchmarking workflow, and motivate various design choices that have been made for the analysis to be effective.

4.1 Project Outline

In order to compare the performance of MySQL, Cassandra, and MongoDB, we require a database system and an appropriate dataset to perform queries on which we can use to compare the performance. The system running the database will be simple for prototyping — a single node and modest resources to be run on our local machines. This will allow us to confirm that the tools are correctly implemented and troubleshoot any issues which may arise. On this system, we have chosen datasets which highlight the strengths of Cassandra and MongoDB. The dataset associated with Cassandra has many fields on which we can run aggregation queries, such as an open IoT dataset or site-tracking data. The dataset associated with MongoDB focuses on the performance benefit obtained on nested data, such as product catalogue data. We anticipate that the dataset choice should highlight the value of the NoSQL solutions and are interested in measuring the performance difference (between Cassandra/MongoDB and MySQL) obtained as the database modelling and system configurations are adjusted. Further, the same dataset will be used to gain insight into measuring the performance difference between the two NoSQL solutions when queried on the same dataset. NoSQL systems are often horizontally scaled so, time permitting, we would like to deepen our insight by scaling our systems to have higher resource allocations and multiple nodes.

The database and benchmarking systems will be setup, run the specified workload queries, and lastly output the raw performance data. The raw data is transformed with scripts to information for comparisons and to graphically understand the data. This output can then be discussed. The comparisons may provide unexpected insight because performance research of NoSQL is not typically performed with modest resources.

4.2 Benchmarking Workflow

Figure [2] 1 graphically outlines the benchmarking process as discussed below. This process is repeated for each testing system required — this includes different databases, the datasets, and the scaling of the system resources (vertically/horizontally).

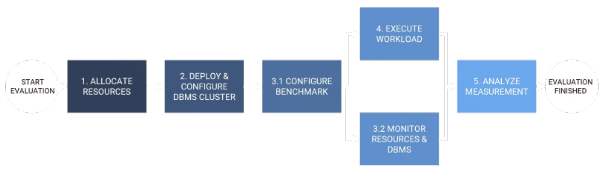

Figure 1: Outline of steps required to benchmark a database

1. Allocate Resources: We setup the server instance or local machine according to the system resources needed for the test. This is adjusted as we scale the resources applied as per the benchmarking required.

2. Deploy and Configure DBMS Cluster: The database being used (MySQL, Cassandra, or MongoDB) is installed and configured.

3. Configure Benchmark: YCSB is setup in this phase.

4. Execute Workload: Our custom workload with the queries being benchmarked is executed. YCSB monitors the resources and DBMS for analysis in the next phase.

5. Analyse Measurements: Our custom scripts for transforming the outputted data are used to assist in comparing the measurements obtained.

4.3 Design Choices

This project undertakes experimental research, so to ensure the integrity of our results, various design concerns need to be disclosed.

The project will be containerized with Docker to ensure that each benchmark run begins from a clean environment. This creates an isolated testing environment. Containers also allow for strict control of system resources to ensure that our benchmarks are only running with the stipulated resources. Further, containers allow for easy portability if we scale the system on the cloud and for test reproducibility.

The benchmarking system we use must be a third-party application to avoid any differences which may occur in the internal benchmarking systems of MySQL, Cassandra and MongoDB. That is, the same benchmarking tool must be used for testing across each database implementation. Further, YCSB avoids exactly modelling particular application(s), as is done in benchmarking systems such as TPC-C [1]. This allows our study to focus on performance with regards to the modelling of our database and the dataset that is being queried.

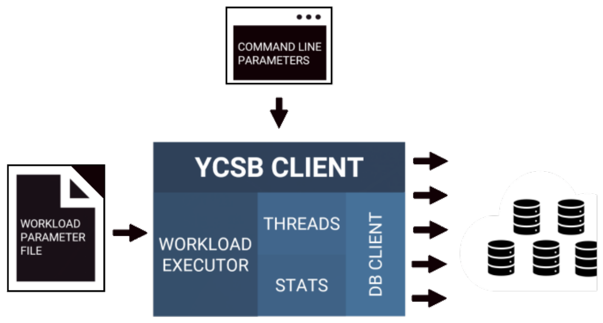

Figure 2: How YCSB works

Figure [3] 2 provides the YCSB structure. The YCSB client receives a workload file specifying the custom queries we intend to run on our dataset and uses the command line parameters to manage YCSB. The 'DB Client' in figure 2 acts as an API for YCSB to communicate with the database which is independently configured for testing.

5 ANTICIPATED OUTCOMES

5.1 System

5.1.1 Software. The proposed work requires us to become proficient in using the following software tools:

- Structured Query Language (SQL)
- MongoDB Query Language (MQL)
- Cassandra Query Language (CQL)
- Yahoo! Cloud Serving Benchmark (YCSB)
- Docker
- Python/R (data comparison scripts)
- Dataset migration scripting (transforming raw public datasets)

We anticipate that each will take some time to become familiar with.

5.1.2 Key queries. Queries will be designed to showcase the strengths and weaknesses of each database within each dataset. For MongoDB, this will mean designing queries that show how powerful embedding can be, and queries that show its limits (where referencing might be better). For Cassandra, queries that read/aggregate a range of columns must be designed. For MySQL, varying the number of joins needed per query will show the database's strengths and weaknesses. Simultaneously, whilst queries which exhibit the strengths

[2]https://benchant.com/blog/ycsb

[3]https://benchant.com/blog/ycsb

and weaknesses of the databases are important for benchmarking, we should also focus on queries that are commonly implemented per dataset use case.

5.1.3 Design challenges. We shall face a number of design challenges in our project. First, we shall need to ensure that the chosen datasets have optimally configured databases. Open datasets are stored without clear structure and will need to be transformed to meet the different data models of each database. This will involve normalising for MySQL, careful choosing between embedding and referencing for MongoDB, and appropriately chosen column families and index keys for Cassandra.

Decisions regarding indexing data will also be challenging. Optimizing indices of a given dataset can have significant implications for read/write speeds. Using indexes on data that is write-heavy can reduce performance due to constant updating of index files. This must be kept in mind as we create indices in our 'read only' experiment. All three databases have a wide range of sophisticated indexing options which must be carefully considered.

Scaling the resources of our database implementations will pose a challenge. This will require server implementations and understanding AWS tools such as EC2 where the researchers are inexperienced. Further, specific database scaling is also of concern — specifically for MySQL because relational databases are known to be difficult to scale horizontally.

5.2 Expected Results and Impact

The two datasets were selected to showcase the strengths of MongoDB and Cassandra. MongoDB's nested structure is expected to provide it with a significant advantage over the other two in product filtering. However, for more complex queries, like those involving multi-collection aggregation, the difference in performance should be less obvious. Since IoT datasets tend to have many columns, Cassandra's columnar data model will be better suited to this dataset than MySQL. Cassandra and MongoDB are designed to eliminate null values and we expect similar performance for simple queries and a similar data size. However, for complex aggregations, as expected to be run on IoT datasets, Cassandra is expected to outperform its competition by increasing margins as horizontal scaling is increased.

Latency and throughput are important for specific use cases. Low latency is crucial in web applications. Conversely, business intelligence systems demand high throughput. Through our research, we aim to unveil the true differences in latency and throughput performance of the three databases for the given datasets. Our results will thus be impactful for system designers.

While many performance studies on Cassandra have looked at the real-time analysis capabilities of the database on Twitter data, less attention has been given to IoT data analysis performance. We aim to fill this gap and support system developer database decision making with regards to IoT data analysis. For the product catalogue dataset, realising the current performance differences between databases will allow companies to assess whether it is worth making the move to a different database because many companies still use relational databases.

5.3 Key Success Factors

In order to provide satisfactory answers to this project's research questions, we need to complete the first three components of the project outline shown in figure 3 (onion diagram). Results collected through completion of these components will allow us to credibly discuss and conclude how each database compares for each dataset. Beyond succeeding in the first three components, reconducting our experiments on scaled architectures (vertically and/or horizontally) would provide even greater depth to our discussions. Achieving a scaled comparison would mean success beyond the requirements of the project.

6 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

Our proposed research contains no significant ethical or legal concerns. As our experiments do not necessitate any human or animal subject testing, and do not disclose any personal information, no ethical clearance is warranted. All datasets used to capture performance indicators are publicly available and, where necessary, will be anonymised in order to protect personal information privacy. All software proposed for use in our research is publicly available or open-source, allowing for free and unrestricted use. All work referenced in this project will be correctly cited to ensure copyright and IP compliance.

The intellectual property for this project belongs to the researchers, Bradley Culligan and David Court, as well as the University of Cape Town, following the standard UCT Intellectual Property Policy.

7 PROJECT PLAN

7.1 Risks

A risk assessment matrix is attached in the Appendix section A. We outline various risks that have been recognised, their impact to the project, the probability of them occurring, as well as strategies we have in place for risk management, mitigation, and measurement.

7.2 Timeline & Milestones

A Gantt chart can be found in the Appendix section B. It contains the timeline for the project along with the project deliverables and milestones. The project timeline can be summarised into the four key stages that prioritise core components first. The onion diagram below (in figure 3) illustrates these key stages.

First, we shall design a simple prototype to be run on our local machines. This provides initial results and ensures that we have successfully setup the databases, YCSB, and appropriate output is obtainable. We then need to transform the raw output data for comparisons so we create scripts to achieve this. We then measure database performance on the other database's chosen dataset for a performance comparison between NoSQL databases. The datasets have been chosen to highlight database strengths so the importance here is to measure the difference in performance to analyse the database's differences in effectiveness. Lastly, time permitting, we increase the resources (vertical/horizontal scaling) allocated to our system.

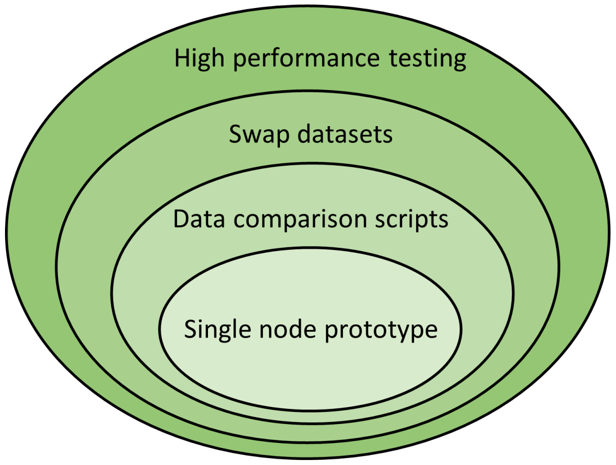

Figure 3: Scope Diagram

7.3 Deliverables

The key deliverables for the project are presented in table 1 below.

Deliverable	Due Date
Proposal Presentation	24 May
Project Proposal	27 May
Revised Project Proposal	15 July
Prototype Demonstration	25-29 July
Final Paper Draft	23 August
Final Paper	2 September
Final Code	5 September
Final Demonstration	19 September
Project Poster	3 October
Project Web Page	10 October

Table 1: Project Deliverables

7.4 Resources Required

Beyond the software tools named in section 5.1.1, other key resources will include:

(1) Open datasets (product catalogue & IoT data)
(2) A local machine on which to run containerised tests
(3) Access to AWS EC2 instances or cluster technology

7.5 Work Allocation

David and Bradley will collaborate equally towards all shared deliverables including the project proposal and presentation, final demonstrations, project poster, and project webpage development. They will work together to understand the intricacies of YCSB and share ideas on how to design meaningful experiments. For their chosen dataset(s), David and Bradley will individually design queries and capture related performances, comparing MySQL performances to MongoDB and Cassandra respectively. They will also swap datasets and measure the other dataset's query performance using their own database. Each partner will be responsible for discussing all results pertaining to their own dataset and database in the final paper.

REFERENCES

[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10* (2010). https://doi.org/10.1145/1807128.1807152
[2] Rick Copeland. 2013. *MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database.* " O'Reilly Media, Inc.".
[3] Alejandro Corbellini, Cristian Mateos, Alejandro Zunino, Daniela Godoy, and Silvia Schiaffino. 2017. Persisting big-data: The nosql landscape. *Information Systems* 63 (2017), 1–23. https://doi.org/10.1016/j.is.2016.07.009
[4] Elif Dede, Bedri Sendir, Pinar Kuzlu, Jessica Hartog, and Madhusudhan Govindaraju. 2013. An evaluation of Cassandra for Hadoop. *2013 IEEE Sixth International Conference on Cloud Computing* (2013). https://doi.org/10.1109/cloud.2013.31
[5] Deka Ganesh Chandra. 2015. Base analysis of nosql database. *Future Generation Computer Systems* 52 (2015), 13–21. https://doi.org/10.1016/j.future.2015.05.003
[6] Venkat N. Gudivada, Dhana Rao, and Vijay V. Raghavan. 2014. NoSQL systems for Big Data Management. *2014 IEEE World Congress on Services* (2014). https://doi.org/10.1109/services.2014.42
[7] Adity Gupta, Swati Tyagi, Nupur Panwar, Shelly Sachdeva, and Upaang Saxena. 2017. NoSQL databases: Critical analysis and comparison. In *2017 International conference on computing and communication technologies for smart nation (IC3TSN)*. IEEE, 293–299.
[8] Jing Han, Haihong E, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *2011 6th International Conference on Pervasive Computing and Applications*. 363–366. https://doi.org/10.1109/ICPCA.2011.6106531
[9] Jing Han, Ee Haihong, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications*. IEEE, 363–366.
[10] Samiya Khan, Xiufeng Liu, Syed Arshad Ali, and Mansaf Alam. 2019. Bivariate, Cluster and Suitability Analysis of NoSQL Solutions for Different Application Areas. (2019). arXiv:arXiv:1911.11181
[11] Pwint Phyu Khine and Zhaoshun Wang. 2019. A review of Polyglot persistence in the big data world. *Information* 10, 4 (2019), 141. https://doi.org/10.3390/info10040141
[12] Rob Kitchin and Gavin McArdle. 2016. What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets. *Big Data & Society* 3, 1 (2016). https://doi.org/10.1177/2053951716631130
[13] Avinash Lakshman and Prashant Malik. 2010. Cassandra. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40. https://doi.org/10.1145/1773912.1773922
[14] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. 2011. NoSQL databases. *Lecture Notes, Stuttgart Media University* 20 (2011), 24.

APPENDIX

A RISK MATRIX

B GANTT CHART

Risk	Probability	Impact (1-5)	Mitigation	Monitoring	Management
Partner dropout	Low	2	Check-in with each other frequently	Check-in on each other's physical and mental health	Ensure work is divided such that remaining partner can submit meaningful project
Unexpected difficulties during experimentation	Medium	3	Create realistic project timeline, including buffer time, and following onion model	Follow project timeline closely and rebuild if necessary	Reduce experiments to core/important components only
Hardware failure	Low	2	Ensure all files are synced to cloud storage	Track hardware performance and load-shedding schedule	Use UCT honours lab PC until hardware is fixed/replaced
Difficulty securing funds for AWS EC2 instances	Low	3	Confirm with supervisor whether department will cover cost	Follow up to ensure costs will be covered	Pay for (short term) service ourselves or abandon scaling
Cluster (EC2) setup and distributed database pose a significant challenge	Medium	2	Cluster scaled analysis is only an extension to deepen insight (does not directly affect our research results)	Analyse difficulties found when setting up cluster	Learn cluster setup and use cluster before project use

Table 2: Risks Matrix

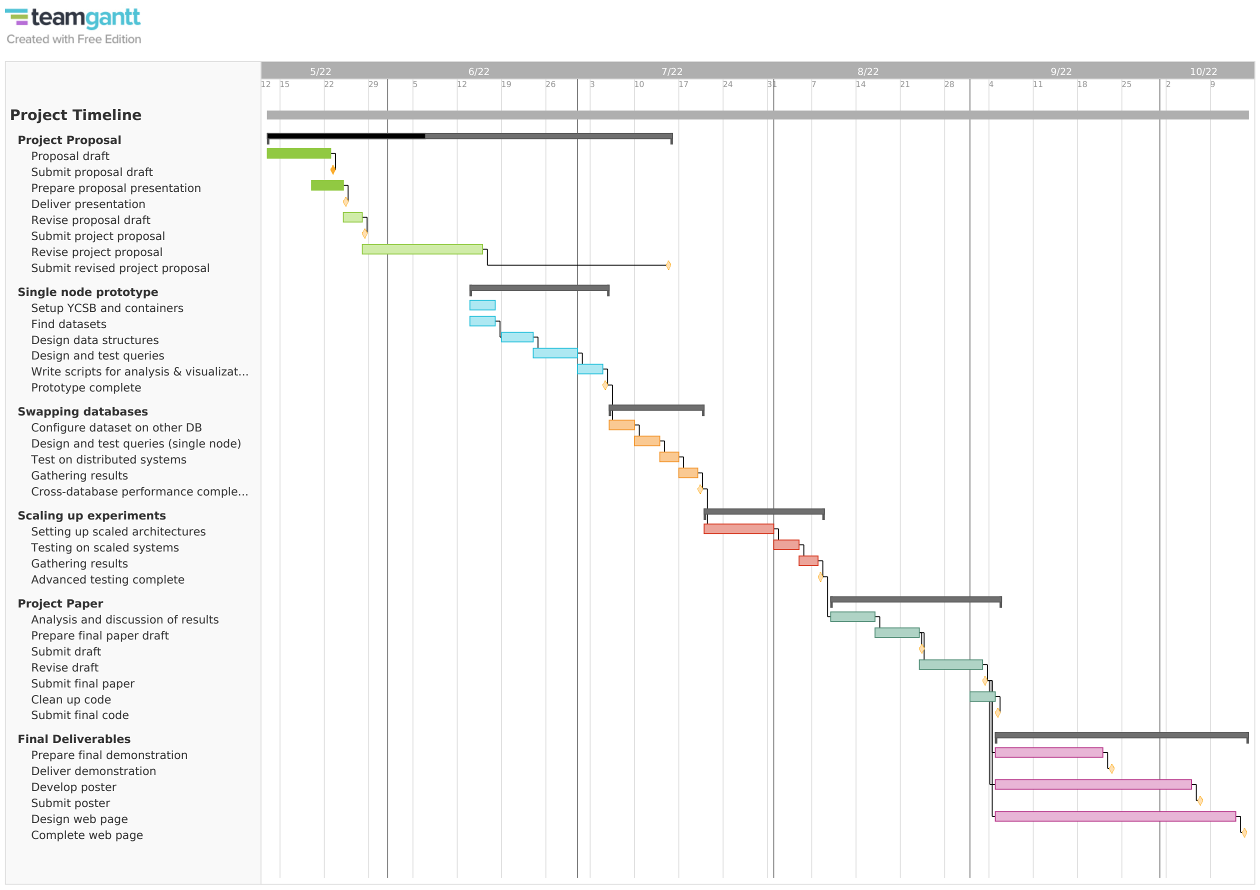

Figure 4: Project Timeline