Distributed Databases for Better Performance, Scalability, Resilience and Transactional Integrity in the case of MOENCO

Hiwot Workneh Denekew
HiLCoE Graduate Programee, Ethiopia
hiworkneh@gmail.com

Abstract

There are several factors to consider when choosing the right database engine for an enterprise, and that depends to a large extent on the type of business the data base is intended to serve. In the case of MOENCO S. Company, which is the main focus of this research, a number of factors are considered, including cost (storage and writing charges in particular), where they value consistency over size, making it incredibly expensive, and (speed over consistency which results in data discrepancy). When selecting a database management system, there will always be tradeoffs to consider, and one size does not fit all. However, there are new database products released that take a lot of inspiration from the NoSQL focus on performance, distribution and scalability and without throwing out all the relational features that have been useful for so many years. These products which include Google's Cloud Spanner and Apache Ignite are sometimes referred to as NewSQL, replacing earlier products such as NoSQL that support ACID transactions instead of SQL queries, and generally following the relational database model while maintaining features oriented towards scaling and speed, much more like the NoSQL products.

The purpose of this paper is to conduct a comparative analysis of cutting-edge database technologies that adopt a multi-model approach and based on the analysis of findings recommend a more viable database solution that is an open source, high performance, distributed database engine, and could serve better the achievement of the organization's objectives while addressing the challenges at hand. This paper also addresses various challenges and benefits in implementing relational and non-relational (NoSQL) databases as well as address the designing factors in Distributed SQL databases like scalability, resilience, and transactional integrity across selected database engines and review of recent work.

Keywords: Distributed SQL, RDBMS, Resilience, Scalability, Geo-distribution, NewSQL

1. Introduction

A database is a logically organized collection of structured data kept electronically in a computer system. A database management system is usually in charge of a database (DBMS). The data, the DBMS, and the applications that go with them are all referred to as a database system, which is commonly simplified to just database. A database engine (sometimes known as a storage engine) is the fundamental software component that allows a database management system (DBMS) to generate, read, update, and delete (CRUD) data from a database. The terms "database engine" and "database server" or "database management system" are commonly interchanged [9].

Databases are a concept that we are all familiar with, and we have all dealt with them; they are difficult to escape if we work or do anything with computing. Databases allow us to operate efficiently while working with enormous amounts of data They make data updates simple and dependable, assist in ensuring accuracy and avoiding redundancy, and provide security measures to manage access to information. There are different kinds of databases engines or database management systems that were intended for different use cases and choosing the right database and the right storage platform to fit the

business needs can shape the success of the entire organization [10].

A database is commonly described as a repository of data or a place to store your information, which is technically correct, but it is not why it is valuable, it is not just because we have data that we need databases, it is for what comes after that and there are major issues and requirements that need to be taken into consideration in this regard such as [11, 12]: what will happen when the data grows, how keeps track of the data when it changes over time, how can our data be shared, fast and made searchable, and accessible, how will the data be reliable, available and credible (Data integrity), how can data be consistent with itself, how can data be protected and secure at all times?

Keeping these requirements in mind, it is critical for every organization to select an appropriate database engine to assist them in dealing not only with a large volume data they need to process and manage but also with all the additional issues that having a well-established and functioning database management system constantly entails. It will be up to the company to determine what is most important the organization considering the requirements discussed above [11].

Databases are one of the few technology products released 30 - 40 years ago that are still relevant today. For instance, IBMs DB2 which was released in 1983 has remained a viable product and a relevant skill ever since. In addition, PostgreSQL, which is a popular open-source DBMS, was first released in the 1980s as was Microsoft SQL Server and Oracle that was released in 1978. Even though these products have been improved, updated, and refined over the years they have not lost their continued relevance for decades. These products and other popular DBMSs were built on the same set of ideas and concepts, and they share a similar approach to what a database should be and belong to a particular type of database management system called relational database management systems (RDBMS). While there are other DBMS types, relational ones have been the most popular and most prevalent databases over the last several decades.

The reason why the relational database management systems (RDBMS) products have stayed relevant for decades is because they are very good at what they do and what they do is perfectly suited for many typical business scenarios. But in the last few years, we have seen various amounts of new products released in this space and many more recent database technologies that were designed to deal with specific situations these standard relational databases were not great at. In order to explore and appreciate the added value of the newer ones it is necessary to compare and contrast them with classic relational databases. For that reason, some of the features of the classic relational databases are addressed in this paper.

The research report will begin by describing the characteristics of classic relational database management systems and why they have remained so popular over time, as well as their limitations. Then it compares and contrasts the current marketplace database solutions including other leading database engines present at the moment. Finally, the research report will recommend a cutting-edge database engine with features drawn from both relational and non-relational approaches.

Hence, the objective of this research paper is to conduct a comparative analysis of cutting-edge database technologies that adopt a multi-modal approach and recommend a better database engine that is suited for improved performance, scalability, flexibility, integrity and to propose a solution that is appropriate for the problem domain.

2. Related Work

Database management systems (DBMS) fall into various broad categories. The most common and widely used DBMS is what is called a relational database management system (RDBMS). Other types of database management systems include Hierarchical DBMS, Network DBMS, Object-Oriented DBMS, including a more recent ones that fall under the category of NoSQL Database systems [6].

The relational database management system has been such a consistent part of computing because the features it provides such as, having strict controlled schemas and ACID transactions [6]. These features while working in a lot of general-purpose business scenarios, may not be ideal for every situation. The same database feature that can be hugely beneficial in one scenario can be detrimental in another. In today's world of Agile development where new features and updates are being released and implemented in cycles of weeks or even days, making continual changes to a database schema can become a challenging task. Beyond that, a couple of decades ago, there were no big data sets and real time web applications and there were no expectations for constant availability and global deployment (i.e., taking a database and replicating it across multiple international data centers) [4]. Over the last several years however, additional database products immerged that were intentionally created to deal with these situations relational databases were not designed to deal with, such as, flexible schemas, massive scale and geographic distribution [1].

A distributed SQL database has three important features. these are, Resilience, Scalability and Geographic distribution [2]. While resilience means failure should not affect the user or there should be an ability to return to a previous state after the occurrence of some event or action which may have changed that state, scalability means adding more nodes in order to do more aggregate processing, i.e., weather it is for a number of queries one is able to handle or total storage volume. Geographic distribution, on the other hand, refers to multizone deployment (i.e., replication of data across multiple servers on different locations) where one is able to service zone failures with low latency.

Distributed SQL has the following five defining characteristics [1,4,6],

1. *It should have an SQL API*: This means developers or users of DBMS such as MySql, PostgreSQL or an Oracle should be able to use their knowledge of SQL development and directly apply it to distributed SQL systems. In such cases, there is no need for learning a completely brand-new language in order to get data in or out of a distributed SQL system.
2. *Distributed Data Storage*: This is a case where data can be inserted easily into the system and depending on the replication factor and the number of nods in the cluster, the system should be intelligent enough to know how to distribute that data so that there will not be any human interaction or involvement.
3. *Strongly Consistent Replication*: This means that whenever data is inserted into the system, data integrity is maintained and it's the data that's not eventually replicated to the other nods where the it needs to reside. This is done in a synchronic and consistent fashion were, regardless of communicating with node 1, node 2 or node 3, there will be a consistent view of data in the system.
4. *Distributed Query Execution*: This means there is no need to figure out where the data is and how to pull it together and get it over to the client among a distributed data across a multitude of nodes in a cluster. Instead, one can easily run a Select, an Insert and an Update query and have the system do the rest of the heavy lifting.
5. *Distributed ACID Transactions* [7]: ACID transaction is a well-known concept in traditional RDBMS

but the only degree of complexity that is added here is that this ACIS transactions have to be aware that one is communicating over different nodes, dealing with data that may or may not be present on specific nodes and being able to roll back these transactions. All these aspects need to be taken into account when dealing with ACID transactions inside a distributed system. Hence, support for distributed ACID transactions means, a distributed SQL database automatically distributes and strongly replicates data so that SQL can be executed against it in a distributed and ACID compliant manner.

With the Internet and cloud computing gaining popularity, new types of applications have evolved, increasing the demand for database technology, like low latency and a high concurrent reading and writing rate, Big data storage and access requirements that are efficient, high scalability and high availability, lower management and operational costs, slow reading and writing, limited capacity as existing relational database cannot support big data in search engine, Big System and Multi-table correlation mechanism which exists in relational database.

A variety of different types of databases have emerged to address the aforementioned needs. Because all these novel databases differ significantly from typical relational databases, they are known as "NoSQL" databases. NoSQL can also be translated as "NOT ONLY SQL" to emphasize the benefit of NoSQL [13].

Any non-relational database is referred to as a NoSQL database, and there are many distinct implementations of NoSQL, including table formats, documents, and graphs. The following are the main benefits of NoSQL: 1) fast data reading and writing; 2) mass storage capability; 3) easy expansion; 4) inexpensive cost

On the grey side, NoSQL has several shortcomings, including lack of support for SQL which is an industry standard, lack of transactions, reports, and other additional capabilities, and the fact that most NoSQL database systems launched in recent years are not mature enough. Following an overview of NoSQL database, below is a comparison of Distributed SQL with NoSQL based on the requirements stated before [14].

Table 1: Comparison of Distributed SQL and NoSQL [6, 14]

6.

Characteristic	**Distributed SQL**	**NoSQL**
Performance	Slightly slower read and/or write performance as it is relational	High performance read and/or write
Scalability	Are designed to scale up vertically for more datasets size.	Datastores are designed to scale out horizontally
Resilience to failure	Master-slave replication, which makes the strongest resiliency power	Peer-to-peer replication
Transactional Integrity	Have natural atomic transactions (based on using ACID)	Provide eventual consistency from BASE (Basically Available, Soft State, Eventual Consistency) properties
Structure	Structured tables to store the data	Semi/Unstructured data collections that are not related to each other

From the literature review made above it can be observed that there are emerging opportunities to link

the relational SQL database with NoSQL – Non Relational database features that would allow product designers to effectively address the growing needs of users in terms of balancing performance, scalability, resilience and transactional integrity.

MOENCO is a subsidiary company of Inchcape PLC, a London based company engaged in global distribution & retail leader in the premium and luxury automotive sectors dealing with more than 12 brands of automotive and machinery.

As a leading global automotive dealer, distributor and retailer in Ethiopia, the firm has a high volume of transactions originating from each of its ?? branches in the country and where all of the distributed branches need to be updated from the main branch (particularly in TOYOTA spare parts sales). Standard DBMS features and requirements such as performance, scalability, transactional integrity and resilience to failure are very important for the company's day to day business management. In addition, the business requires high synchronization (quick update and delete statements are required to ensure that data integrity is maintained) and high-speed to adequately respond to the needs of its large and ever-growing customer base.

MOENCO had been using a locally developed tailor-made ERP system called Hillmark which has served the main office and Its branches for over a decade with a distributed system architecture - traditional relational database (Microsoft SQL Server2008-2012). Through time however, the company began to experience significant storage expenses as well as speed concerns as the volume of its data grew. As of the recent March 2019 to be precise, the company switched to a K-ISAM flat-file non-relational database (ADP Autoline Database Filesystem) called CDK Autoline DMS, which is faster than the earlier versions but still presents several challenges. Among these challenges include, data redundancy, requirement of several entries, the fact that specific data is erased when the system archives, cases where transactions are lost owing to lack of data harmonization, and common occurrence of read/write conflicts, to name a few.

However, there are new database technology products released, similar to Google's Cloud Spanner and Apache Ignite that include both features (hybrid databases) and are therefore ideal database engine in the context of MOENCO's requirements. The company requires speed because it's sales (large automotive and after-sales transactions) are dependent on it. It also requires data consistency because the company must perform analytics, study customer data and behavior in order to increase sales, as well as provide data to the government and report to its parent company. This research paper aims to conduct a comparative analysis of cutting-edge database technologies that adopt a multi-model approach with a view to recommend improved and state-of-the art technology solutions that would allow the company to meet its business objectives and meet the ever growing demands of its customers.

3. Comparative Analysis and Recommendation

When it comes to selecting the ideal database for an enterprise, there are several factors to consider. These include different database options that could generally fall into broad data categories, such as Relational, Documented, Key-value, Graph, In-Memory, search, time series, and many more others [7]. The comparative analysis here will focus on the following key database technologies, where each of these database technologies are built for a specific purpose and functionality, making them somewhat unique and different in their own way.

MySQL is a multi-threaded, multi-user SQL (Structured Query Language) database server that is exceptionally dependable [6]. It is intended for use in mission-critical, high-volume production systems, as well as in widely dispersed applications. MySQL has a number of downsides and limitations, including

licensing and proprietary features. It is a dual-licensed software, which exists in a range of commercially available versions under proprietary licenses. As a result, several functionalities and plugins are only available in the paid editions.

Amazon Aurora has been generally available since 2015 as stated on and is based on a proprietary distributed storage engine that replicates 6 copies of data across 3 availability zones for high availability [1]. It is compatible with both PostgreSQL and MySQL in terms of API. Aurora runs in a single-master configuration by default, which means that only one node may process write requests and the rest are read-only replicas. Multi-master configuration is a new feature in Aurora MySQL (although not yet in Aurora PostgreSQL) that allows you to scale writes by adding a second writer node. However, because both nodes now have all of the data, concurrent writes to the same data on both nodes might cause write conflicts and deadlock problems, which the application must deal with. The inability to scale beyond the original two writer nodes, as well as the lack of geo-distributed writes across several regions, are just two of the many drawbacks [1]

PostgreSQL is a powerful object-relational database management system that includes transactions, foreign keys, subqueries, triggers, user-defined types, and functions. PostgreSQL creates a new process for each new client connection [1]. Each new process is given about 10MB of memory, which can quickly add up when dealing with large databases with many connections. As a result, PostgreSQL is usually slower than other RDBMSs, such as MySQL, for simple read-intensive operations [2], which makes low memory performance its major limitation.

Spanner is Google's worldwide distributed SQL database that runs several of the company's mission-critical services, including AdWords, Apps, Gmail, and others. Since 2007, Google has been developing Spanner, initially as a transactional key-value store and then as a SQL database. In 2017 [3], Google Cloud Spanner, a private managed service, made a part of the Spanner system publicly available on the Google Cloud Platform. Cloud Spanner has its own SQL API that isn't compatible with any other open-source SQL database and doesn't support relational data modeling components like foreign key constraints [3], which appears an advantage from Google’s perspective but can be considered as a limitation from users’ perspective.

The sharding, replication, and distributed transaction features of CockroachDB, which is a PostgreSQL-compatible distributed database, is inspired by Google Spanner. Because the API layer is a reimplementation of the PostgreSQL query layer, functional depth is lost. Partial indexes, stored procedures, and triggers, for example, are not supported. CockroachDB also recently abandoned its open-source roots by relicensing the primary database under the commercial BSL 1.0 license, which restricts freedom of usage. Finally, complex functionality such as distributed backups and change data capturing are not available in the premium edition [5] that could prove to be costly to users.

YugabyteDB was inspired by both relational SQL concepts with the Non-relational- NoSQL concepts by using state of the art technologies in both aspects (PostgreSQL and Google Spanner) by considering the fact that PostgreSQL is more popular and fast-growing DB than any other database, and Google spanner is a truly, horizontally scalable and strongly consistent relational database that is cloud native [2]. When looking at the two sides however, the PostgreSQL is not cloud native meaning it cannot do scale, high availability in spite of failures, geographic distribution, distributed transactions etc. Google spanner does not have all the RDBMS features set and the quality that PostgreSQL has but putting these two together is the best way to build the perfect distributed SQL database which is Yugabyte DB.

The main difference between monolithic, single node databases like MySQL or PostgreSQL and

distributed SQL databases like Yugabyte DB is that by default in a monolithic single node database, all writes need to be served from a single node. In order to get more scale from databases like MySQL or PostgreSQL, one has to add more memory more CPU and more storage capacity, in other words 'Scaling Up'. On the other hand, with distributed multi-node SQL databases like Yugabyte DB, writes can be served from any node and in order to get more scale from a distributed database one needs to add more nodes to the clusters, in other words 'Scaling Out' [2]. With that definition in mind Yugabyte DB is built to be a distributed SQL database with a focus on high performance which means the ability to do relatively low latency serving and ability to deal with large amount of CPU, memory, Ram etc. It is an OLTP database and its cloud native so it will run anywhere and doesn't have external dependencies and the database itself is open source just like PostgreSQL.

Table 2: Comparison of PostgreSQL and Google Spanner

Characteristic	**PostgreSQL**	**Google Spanner**	**Yugabyte DB**
Query Layer	Very well known	New SQL flavor	Reuses PostgreSQL
SQL Feature Depth	High (Many advanced features)	Low (Basic features missing)	High (Supports most Postgres features)
Scalability	No Horizontally scalable	Horizontally scalable	Horizontally scalable
Replication	Async	Sync, Reads replicas	Sync, Reads replicas

As shown in Table 2 above, when comparing the query layer and the SQL features we observed as a part of PostgreSQL are well understood, adopted, known and really advanced. On the other hand, Google spanner misses triggers or stored procedures or portal partial indexes a whole bunch of long list of features and has a very basic feature set and it's a completely new flavor of SQL. When we look at what Google Spanner has done though, it has high availability, horizontal scale and distributed transactions. It also does Synchronous replication while PostgreSQL and most of the RDBMS do Asynchronous replication. So with Yugabyte DB which it a completely open source database just like PostgreSQL, has a combination of the best of those two worlds (the best of PostgreSQL and Google Spanner) [2].

PostgreSQL is a single-node open-source RDBMS, that is widely adopted for its powerful set of features while being extensible [2]. However, it is difficult to run PostgreSQL as a cloud-native database to inherently survive failures, by highly available, scale horizontally, and be deployed in geo-distributed configurations. Google Spanner is a distributed SQL database that has these features, however, does not offer the power of PostgreSQL. Combining the best of these two databases would result in a very compelling database. YugabyteDB is a fully open-source distributed SQL database aimed at achieving exactly this goal [1].

Table 3: Comparison of Amazon Aurora, Google Spanner and Yugabyte DB

Feature	**Amazon Aurora**	**Google Spanner**	**Yugabyte DB**
SQL Wire Protocol	Yes (PostgreSQL, MySQL)	No (New SQL flavor)	Yes (Reuses PostgreSQL)
RDBMS Feature Support	High (Many advanced features)	Low (Many features missing)	High (Supports most Postgres features)

Fault Tolerance	Yes (Multi zone/ multi region cluster)	Yes	Yes (maximum number of node failures it can survive while continuing to preserve correctness of data.)
Scalability	Writing is not horizontally scalable (vertically extend all write nodes or master nodes to extend the write throughput)	Horizontally scalable	Horizontally scalable
Transaction concepts	ACID compliant	ACID compliant	Distributed ACID with Serializable & Snapshot Isolation. Inspired by Google Spanner architecture.
Replication	Async	Sync, Reads replicas	Sync, Reads replicas

As shown on Table 3 above, Amazon Aurora reuses PostgreSQL and MySQL as its query layer but Google spanner builds something new from scratch and the feature depth that Amazon Aurora has are much high and supports the RDBMS features than Google Spanner and Yugabyte DB has a combination of both.

From the perspective of cost and speed, which are the two main requirements for efficiency, effectiveness and profitability in MOENCO S.Co. the use of Yugabyte DB is recommended both wither to be deployed on premises or on any cloud platform. The fact that Yugabyte DB supports advanced RDBMS features like triggers, stored procedures, foreign keys and some Postgres extensions and all distributed transactions are guaranteed Atomicity, Consistency, Isolation and Durability, in other words ACID transactions [7], makes it even more attractive as a viable database product.

Table 4: Comparison of Distributed SQL Databases

Characteristics	**Amazon Aurora**	**Google Cloud Spanner**	**Cockroach DB**	**Yugabyte DB**
SQL & Transactions Compatibility	PostgreSQL and MySQL (Full Compatibility)	Proprietary SQL (No Foreign Keys)	PostgreSQL (No Partial indexes, Stored Procedures and Triggers)	Reuses PostgreSQL (Full Compatibility)
Native Failover/ Repair	Automated Repair	Automated Repair	Automated Repair	Automatic repair of missing data after failures, using unaffected replicas as sources
Horizontal Write Scalability	Yes (Multi-Master)	Yes (Auto-Sharded)	Yes (Auto-Sharded)	Yes (Auto-Sharded)
Geographic Data Distribution	Yes (Only a single region can take writes)	Yes (Global clock sync for consistency)	Yes (Global clock sync for consistency)	Yes (Global clock sync for consistency)
Cloud Neutral	No (Proprietary to AWS)	No (Proprietary to Google Cloud)	Yes (Can run in any cloud platform)	Yes (Can run in any cloud platform)
Open Source	No	No	No	Yes (Apache 2.0)

Shown in Table 4 above is a more detailed comparison of the database technologies under investigation.

4. Conclusion

When deciding on the right DBMS, there are now more options than ever and with more products are starting to adopt a multi-model approach [4]. While this paper compares the features and functionalities of these recent database technologies, it does not go in to further addressing the detail technical specifications. According to the comparative analysis Yugabyte DB shows to be feature reach with it being cloud neutral (multi-cloud) and open source as well as its fault tolerance, speed and scaling up and down without incurring down time. The traditional RDBMSs can support ACID transactions, but they cannot deliver distributed transactions or global data distribution without complicated replication architectures and Yugabyte DB incorporates these features in one [1]. But in terms of where we are in using cutting edge technology in our country Ethiopia, one of the things that also needs to be considered is not just the pure feature set of any database product or the cost of it, but also the skills on the IT team.

References

[1] Sid Choudhury, "What is Distributed SQL?", https://blog.yugabyte.com/what-is-distributed-sql/, Last accessed on November 25, 2019.

[2] Galić, Z. and Vuzem, M., "A Generic and Extensible Core and Prototype of Consistent, Distributed, and Resilient LIS". ISPRS International Journal of Geo-Information, 9(7), p.437. 2020.

[3] Wei, Jinliang. "Spanner: Google's Globally-Distributed Database." (2013)..

[4] Velmurugan, L., and S. Manoharan. "Designing Factors of Distributed Database System: A Review." Data Mining and Knowledge Engineering 12.1 (2020): 7-10.

[5] Taft, Rebecca, et al. "Cockroachdb: The resilient geo-distributed sql database." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

[6] Binani, Sneha, Ajinkya Gutti, and Shivam Upadhyay. "SQL vs. NoSQL vs. NewSQL-A comparative study." database 6.1 (2016): 1-4..

[7] Orlunwo Placida, O, and Prince Oghenekaro ASAGBA. "DISTRIBUTED DATABASE MANAGEMENT SYSTEM (DBMS) ARCHITECTURES AND DISTRIBUTED DATA INDEPENDENCE." (2021).

[8] GALIĆ, ZDRAVKO. "LIS IN THE ERA OF BDMS, DISTRIBUTED AND CLOUD COMPUTING: IS IT TIME FOR A COMPLETE REDESIGN?" (2020).

[9] "What is a database?" May 2021, [Accessed 9 December 2021]. [online] Available at: https://www.oracle.com/database/what-is-database.

[10] Watt, A., Chapter 6 Classification of Database Management Systems, 2013. [Accessed 7 December 2021]. [online] Opentextbc.ca. Available at: https://opentextbc.ca/dbdesign01/chapter/chapter-6-classification-of-database-systems.

[11] Battson, D., "Top 10 considerations when choosing a Database Management system" June 2014, [Accessed 18 October 2021]. [online] Available at: https://datahq.co.uk/knowledge-hub/blog/top-10-considerations-when-choosing-a-database-management-system.

[12] Weiss, Christine. "Selecting a Database Management System (DBMS)–A Practical and Quasi-Technical Analysis of Relational Database Management Systems (RDBMS) and Object Database Management Systems (ODBMS)." University of Colorado at Colorado Springs (2012).

[13] Han, Jing, et al. "Survey on NoSQL database." 2011 6th international conference on pervasive computing and applications, pp. 363-366. IEEE, 2011.

[14] Al Hinai, A. H. "A Performance Comparison of SQL and NoSQL Databases for Large Scale Analysis of Persistent Logs.", 2016.