Distributed PostgreSQL

Pranav Chatur
Contact No: +91 87999 65893
Email: pranavchatur@outlook.com

Abstract

Databases have become ubiquitous in modern web-based applications. PostgreSQL, a widely used relational SQL database proves to be a great choice but lacks horizontal scalability, leading developers to favor distributed NoSQL databases instead. The implication of Brewer's CAP theorem is that no database can achieve consistency, availability and partition tolerance simultaneously. Apache Cassandra is a popular NoSQL columnar database which guarantees partition tolerance and availability. Despite its distinct architectural philosophy, the schemaful nature of Cassandra serves as a valuable source of design insights for PostgreSQL. By making subtle adjustments to certain aspects of Cassandra's approach, it is possible to integrate these insights into PostgreSQL and thereby challenge the CAP theorem. The paper re-imagines the capabilities of a relational database by introducing sharding of a database. By opting for a row-wise data distribution in PostgreSQL, horizontal scalability can be achieved while maintaining the database's relational structure, though it may involve a trade-off in availability. The paper briefly mentions distributed atomic transactions, the key to ensure consistency in such a database. The paper ends with mentioning Citus, an open source extension to Postgres and a few other NewSQL databases which have adopted a similar approach as mentioned in the paper.

1 Background

A database is an organised collection of data and a database management system (DBMS) is the software that is used to query, insert and manipulate the data present in the database. Often coupled with a friendly graphical user interface (GUI), they serve as a way to store and serve data to the users as and when required in the desired form.

Like in case of every nuanced system, there is no one right way to create a database management system. The choice of design paradigm depends on the specific requirements of the application, and different paradigms offer unique advantages and trade-offs that make them suitable for different scenarios. This paper focuses on three desirable traits of a database system: Consistency (in terms of data retrieval), Availability (accessibility of stored data) and Partition Tolerance (tolerance to handle distributed data). However limited, these three virtues of databases can discern many of the databases into categories of their own.

Among the many available database softwares, the paper primarily considers *PostgreSQL* [5], an open-source relational SQL database. A *relational database* is a database which is based on the relationships between different macro-entities present in the database and *Structured Query Language (SQL)* is a programming language used to perform operations on the records present in the database. PostgreSQL or Postgres, is one of the most widely used database softwares as it guarantees atomic transactions while simultaneously maintaining consistency, isolation and durability of the database (collectively called as *ACID properties* [7].

Of the earlier mentioned three virtues (Consistency, Availability and Partition Tolerance), Postgres, by the very nature of its design, only supports Consistency and Availability. In fact, *Brewer's theorem* [2] states that a database can satisfy at most, only two of the three characteristics simultaneously.

Partition Tolerance is the ability of a database to continue operating despite an arbitrary number of messages being dropped (or delayed) by the network between nodes. This property is inherently satisfied by distributed databases. Therefore all distributed databases decide to either support consistency or availability, but cannot cater to both simultaneously.

The purpose of this paper is to challenge Brewer's CAP theorem. By exploiting the architecture of Postgres, the aim is to make it behave like a distributed database.

2 Case Evaluation

This section briefly talks about characteristics of distributed databases, what makes them a popular choice over traditional relational SQL databases and explores a popular NoSQL database, *Apache Cassandra* [4]. The section ends with projecting Cassandra's design strategy onto Postgres for it to get the best of both SQL and NoSQL.

2.1 Characteristics of distributed databases

A Distributed Database Management System (DDBMS) [3] consists of a single logical database that is split into a number of fragments, with each fragment being an autonomous entity and capable of communicating with other fragments. Each fragment is generally assigned part of the complete database. This process is called *sharding* and the part of the database is called a shard. Distributed databases are typically NoSQL databases, i.e. non-relational highly scalable databases with a query language of their own.

One of the reasons NoSQL databases are preferred over SQL databases is their distributed nature. SQL databases, by their design, are only vertically scalable, i.e. existing nodes can be provided with more computing power. Even so, over time the cost of scalability increases significantly in case of SQL databases. On the contrary, NoSQL databases can be scaled both horizontally and vertically making them a better long-term alternative.

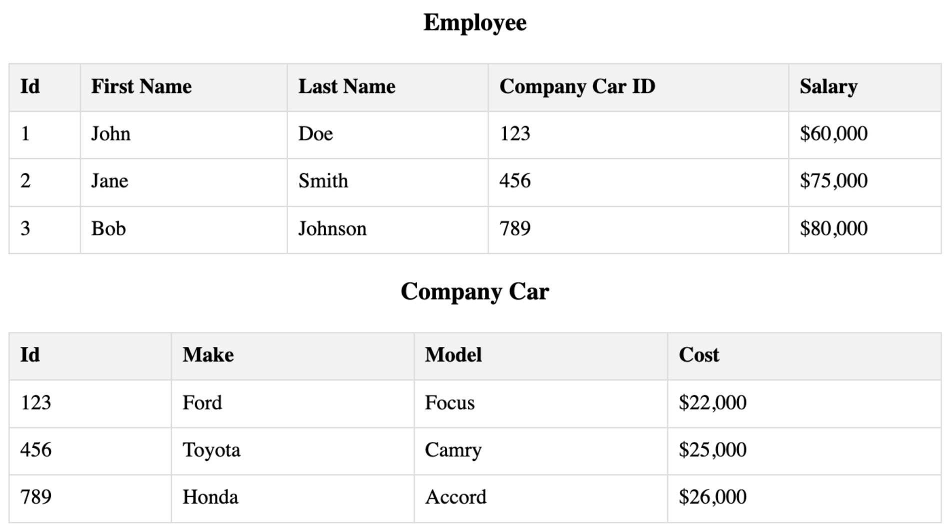
The power to scale horizontally can only be brought about when SQL adopts some form of distributed architecture. Therefore there is a strong reason to look into how some of the NoSQL databases handle distributed data.

2.2 Apache Cassandra: NoSQL columnar database

Apache Cassandra is an open source NoSQL distributed database. It is tacitly partition tolerant and prioritises high availability over consistency. However, Cassandra offers tunable consistency, giving developers a choice between eventual consistency and strong consistency based on the needs.

What makes Cassandra an interesting NoSQL database for the purpose of this paper is its architecture. Unlike most NoSQL databases, Cassandra uses a SQL-like language CQL and isn't schema-less, i.e. it is schemaful. Developers are expected to store the data in tables and are required to specify schema for each table, very similar to relational SQL databases. This design paradigm effectively bridges the gap between SQL and NoSQL databases, offering the advantages of both.

How then does Cassandra distribute its data? It does so by distributing the different columns of a table across different nodes. Additionally, Cassandra, by design, doesn't allow for joins and thus adopts a Query-First-Approach. Instead of a Data-Model-First approach, where the data schema and structure are defined before queries are designed, this approach empha-



Employee

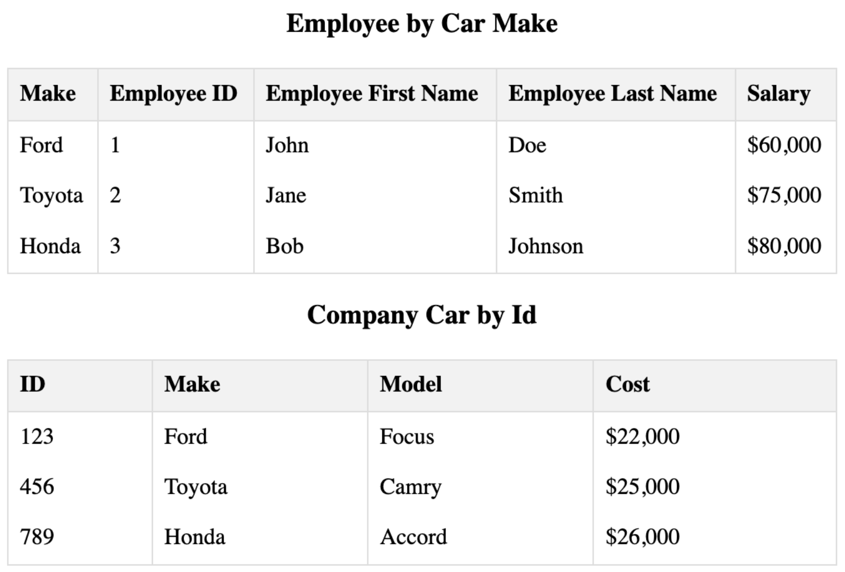
Id	First Name	Last Name	Company Car ID	Salary
1	John	Doe	123	$60,000
2	Jane	Smith	456	$75,000
3	Bob	Johnson	789	$80,000

Company Car

Id	Make	Model	Cost
123	Ford	Focus	$22,000
456	Toyota	Camry	$25,000
789	Honda	Accord	$26,000

Figure 1: Data-Model First Approach

sizes creating a well-defined schema and then building queries to operate on that schema. Figures 1 and 2 depict the how data is arranged differently by relational databases and Cassandra. In Figure 2, the tables are purposefully structured to accommodate specific queries, namely, 'Get Employee by Car Make' and 'Get Company Car by Id'.



Employee by Car Make

Make	Employee ID	Employee First Name	Employee Last Name	Salary
Ford	1	John	Doe	$60,000
Toyota	2	Jane	Smith	$75,000
Honda	3	Bob	Johnson	$80,000

Company Car by Id

ID	Make	Model	Cost
123	Ford	Focus	$22,000
456	Toyota	Camry	$25,000
789	Honda	Accord	$26,000

Figure 2: Query First Approach

The core strength of Cassandra lies in its linear scalability, with the ability to scale from 8 to 800 nodes and back to 8 nodes with minimal downtime while still supporting extremely heavy write-loads. This makes it a strong choice for sensor-networks, real-time analytics and

IoT applications.

2.3 Extention to Postgres

The natural question which arises is whether the exact same approach can be adopted for Postgres or not. Sadly no. Postgres, as a matter of fact, all relational databases use a row-wise data storage model. A relational database must ensure that rows with related data are stored together. Distributing different columns to different nodes of Postgres would destroy the relational nature of the database. The idea is to come up with a relational database that supports still supports data distribution.

3 Proposed Solutions

There is some need of clarification and decoupling required to understand and adopt the features of distributed systems to relational databases. Table 1 highlights the specifications of our desired distributed SQL database. Therefore, the objective is to achieve a relational database that supports horizontal scaling.

Feature	**Monolithic SQL**	**NoSQL**	**Distributed SQL**
ACID Compliant	Always	Not necessarily	Always
Scalability	Vertical	Horizontal and Vertical	Horizontal and Vertical
Consistency	Always	Not necessarily	Always
CAP properties	C and P	P and trade-off(C or A)	C and trade-off(A or P)

Table 1: Monolithic SQL vs NoSQL vs Distributed SQL

Consequently, the logical progression from Cassandra's columnar approach to PostgreSQL would be to distribute the data row-wise across various nodes.

3.1 Distributed Atomic Transactions

The idea would be to, first and foremost, maintain consistency across the different rows located in different nodes. This can be ensured by updating all rows simultaneously or none at all, i.e. distributed atomic transactions. Distributed transactions would be the transactions spanning multiple shards which would be treated as one single atomic transaction.

3.2 Sharding rows for horizontal scalability

Further, horizontal scalability is achieved by distributing rows to different nodes which form a distributed Postgres ecosystem. Depending on the number of nodes present the load can be distributed evenly or the traditional master-slave architecture can be maintained across different nodes as well. As the number of records grow, they can be distributed across nodes and in case of addition of a new node to the ecosystem, the load of all nodes in the system can be appropriately reduced effectively. Therefore the SQL database achieves horizontal scalability to quite an extent.

3.3 The trade-off

The database would be mostly available except during the times of distributed atomic transactions and during the additions and/or removal of nodes from the distributed environment. In theory, this is where the trade-off between availability and partition tolerance lies. Therefore, similar to Cassandra's tunable consistency, a distributed Postgres would offer a tunable partition tolerance.

Ultimately, it must be realised that Brewer's CAP theorem is a rather simplistic and a broad generalisation of classification of database systems. In practice the spectrum would be much more nuanced than just Consistency, Availability and Partition Tolerance. There are many other factors which have an impact on both, user and developer experiences, but the CAP theorem fails to capture them, like latency, concurrency, durability, maintainability, functionality, operational simplicity and financial feasibility.

4 Conclusion

The purpose of the paper was to re-evaluate the capabilities of a Relational Database Management System (RDBMS) and stretch them to achieve a different class of databases. The paper considered PostgreSQL, a widely used SQL database and contrasted its working with a popular NoSQL columnar database, Apache Cassandra. Further it looked into how Postgres could benefit by subtly modifying the design decisions of Cassandra. It proposed a viable, albeit highly abstracted and theoretical, way to achieve horizontal scaling in relational SQL databases by the means of distributing the Postgres database row-wise across the different nodes which form a distributed ecosystem. The paper ends with briefly mentioning database systems existing in present day which are designed on the concepts mentioned in the paper.

5 Existing technologies

5.1 Citus: Postgres extension

The Citus database is an open source extension to Postgres. Instead of being a fork, it extends Postgres by providing additional features like sharding, distributed tables, reference tables, a distributed query engine, columnar storage and the ability to query from any node. Therefore, by using this extension, not only can the distributed features be used but users can also leverage the latest Postgres features, tooling, and ecosystem.

In [1], the authors go in detail of Citus. The paper starts with discussing potential use cases which benefit from a distributed relational database. It then delineates the architecture of Citus which is similar to Postgres' master-slave model. It also explains the other extensions like distributed query planner and introduces distributed transactions. It ends with elucidating the performance of Citus on an actual system.

5.2 NewSQL: Distributed SQL databases

The advent of distributed systems can be seen in the field of databases quite distinctly. Distributed Relational Database Managements Systems like Google Spanner, YugaByteDB, CockroachDB and NuoDB and have become popular choices for a database. On a gross level, all these databases are consistent and partition tolerant and claim to be available for all practical purposes.

- Google Cloud Spanner is a globally distributed, highly available database that provides the unique capability of strong, externally consistent transactions across the globe. Its selling point is the ability to effortlessly scale to meet the demands of global applications while ensuring data accuracy and reliability, all within a fully managed cloud service. However, it is a closed-source database and available only on Google's Cloud Platform.

- YugabyteDB is an open-source solution for Postgres with the primary objective to achieve infinite scalability in cloud environments. Though this can be complex in a cloud-native settings, what sets YugaByte apart is its ability to facilitate a multi-cloud strategy, which enables users to host databases across multiple cloud platforms simultaneously. This approach effectively eliminates the concerns associated with vendor lock-in, providing greater flexibility and freedom of choice in cloud hosting.

- CockroachDB [6], an open-source database developed by ex-Google employees, offers compatibility with PostgreSQL but was purposefully constructed from the ground up using Go to achieve horizontal scalability in cloud environments.

The dominance of such databases has made the community feel the need to define a new class of databases, called NewSQL, as opposed to SQL or NoSQL. NewSQL databases are a category of relational databases that aim to combine the strengths of traditional SQL databases (such as strong consistency and transactional capabilities) with the scalability and high availability features of NoSQL databases.

References

[1] Umur Cubukcu, Ozgun Erdogan, Sumedh Pathak, Sudhakar Sannakkayala, and Marco Slot. Citus: Distributed postgresql for data-intensive applications. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2490–2502, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[3] Swati Gupta, Kuntal Saroha, Bhawna, M Tech, Pdmce Scholar, and Bahadurgarh. Fundamental research of distributed database. *International Journal of Computer Science and Management Studies*, 11, 08 2011.

[4] Malik P. Lakshman, A. Lakshman, a., malik, p. (2009). cassandra: A decentralized structured storage system. in proceedings of the 5th international workshop on large scale distributed systems and middleware (ladis). In *Proceedings of the 5th International Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2009.

[5] Michael Stonebraker, Lawrence Rowe, Karl Hirohama, Donna Hsiao, André Kerne, Bruce Lindsay, Denny Quass, and Danny Schuh. The implementation of postgres. *ACM SIGMOD Record*, 17(1):204–215, 1988.

[6] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Cockroach Labs, Inc., 2020.

[7] Gottfried Vossen. *ACID Properties*, pages 19–21. Springer US, Boston, MA, 2009.