Comparison of SQL Relational Databases and NoSQL Graph Databases

Teodor Fratiloiu

***Abstract*—Classic, relational, SQL databases have proven their reliability, and versatility, in many different scenarios, over the past few decades. However wide their field of application may be though, there are still many situations, where one can be limited, by the constraints, of these traditional data structures. Efficiently querying very large user databases is one such situation. In this work, we are going to investigate a different paradigm for big data management: Graph Databases. This relatively new breed of data management systems are already finding wide acceptance, and are being actively deployed across many industries. Use cases such as social media applications come to mind, where scalability is of prime importance, since datasets could theoretically achieve infinite dimensions. In this work, we will compare Graph, NoSQL databases, their strengths, and weaknesses, to classic, relational SQL databases.**

Keywords: Relational Database, NoSQL Graph Database, Tablespace

I. Introduction

One may commence by presenting the main issue with classic, SQL databases. It is the exponential way in which hardware requirements scale up with the amount of data stored. This means that it becomes exponentially more challenging to maintain speed, RAM and storage amounts within reasonable constraints, as the amount of data increases.

Let us therefore introduce NoSQL databases. The main difference between these, and SQL databases, is how they scale. While SQL databases scale *vertically*, NoSQL databases scale *horizontally*. Specifically, it is the BASE (Basically Available State and Eventually Consistent) property of NoSQL databases which enables them to be a perfect choice for very large, connected, amounts of data.

For the purpose of this paper, we shall consider the term *relational* database as synonymous to *SQL* database, and the term *NoSQL* database as equivalent to *Graph* database, unless other specified. As such, the respective pairs of terms will be used interchangeably.

It has been proved in research literature, as well as by the industry, that Graph databases perform radically better than relational databases for big data applications. This is especially true in scenarios where one can use the connected nature of data to their advantage. One prominent example of this is how Neo4j (NoSQL) performs much better than Oracle 11g (SQL).

However, let us not forget that Graph databases are not the only kind of NoSQL database out there. In fact, there are four types in total. Let us therefore examine these in detail.

A. Types of NoSQL Databases

There are four different kinds of NoSQL databases. Their properties and use scenarios depend on the needs and requirements of the applications in which they are used.

TABLE I
Types of NoSQL Databases with Examples

Type	Package	Query Language	Property Support
Graph	Neo4j	CypherQL	ACID
Graph	ArangoDB	AQL	
	Hive	HQL	
Graph	OrientDB	SQL	ACID
Key-Value	Riak		
Key-Value	Redis		
Key-Value	Voldemort		
Column	Hbase		
Column	Accumulo		
Document	MongoDB		
Document	CouchDB		

- **Key-Value**: these are, essentially, hash tables, where only unique hash keys are used to search for and point to specific values; the contents themselves are not used during queries.
- **Document**: Data is stored as a key/value tuple, and both the content, as well as the *key* are used for searches, which constitutes a major difference from the type above. Documents of this type can be stored in XML, JSON and BSON formats, and popular apps using this type of database include MongoDB and CouchDB.
- **Wide-Column**: These databases are made unique by the hybrid approach used to store data. In this scenario, which is mainly used in *cluster environments* and for distributed data, we will combine the features of relational databases with a key-value storage schema. Examples such as Hbase, Cassandra and Accumulo spring to mind.
- **Graph**: This type is used where the data has a high degree of connectivity, since such databases can naturally store relationships between Independent pieces of data. Examples include Neo4j, Pregel, ArrangoDB and OrientDB.

Actual examples, together with actual package names, query languages and properties, can be found in *Table I*.

B. Relational Database Design Optimization

Let us look into how the performance of classic databases can be optimized. To this end, a process called *tuning* can be involved. For instance, to improve the query performance of *Oracle 11g* tables, we can use different tablespaces for each individual schema. This results in more than one data file used for storage, which in turn allows us to optimize the size of memory blocks used. This is useful, since both small and large memory block sizes can be desirable, depending on

the scenario. For example, *warehouse type* data requires large memory blocks, whereas for *OLTP* type data, we are required to use small block sizes. The total space allocation in a file is defined as an *extent*, whereas one unit of I/O data is referred to as a *block*. The ACID (Atomicity, Consistency Isolation, Durability) property of SQL databases helps maintain consistency across the data, and enables efficient polling and querying, but even this is eventually overcome by the sheer size of modern databases for online, user-generated content.

Even if SQL databases store data in separate tables, and normalization techniques, such as 1NF, 2NF and 3NF, can be used to reduce redundancy, this method remains unsuitable for modern applications. For instance, although it is possible to *join* queries when searching a SQL databases for entries, this too becomes inefficient when using 20 or more parameters.

II. State of the Art

In this section, we shall look into, and compare, the different properties of SQL and NoSQL databases. For starters, let us revisit the scalability issue. As previously discussed, the more varied and larger the amount of data present in a SQL database, the more CPU and RAM a system will need. This is because relational databases scale *vertically*, in contrast to NoSQL databases, which scale *horizontally*. SQL databases are traditionally a good fit for applications where data integrity and security are important, since cryptolographic procedures and information security can be more easily implemented in such a scenario. NoSQL databases, however, are a great fit for very large, and growing, datasets, because they allow us to, for instance, add extra servers to an existing system with minimal intervention.

III. Methods and Contributions

Let us, for this section, start by introducing Eric Brewer's *CAP Theorem* - Consistency, Availability and Partition Tolerance. A term by term analysis looks as follows: *Consistency* describes the property of a system of achieving a *stable state*, after one (or more) *write* operations. *Availability* describes data, namely it being present and ready for *read* operations at one desired spot, after having been modified (right there or somewhere else). *Partition Tolerance* refers to a system's ability to continue functioning, even if its resources are spread over different locations and nodes. Relational databases traditionally possess all 3 properties, whereas NoSQL databases (depending on the implementation) customarily only use the *A* and *P* parts.

In relational databases, primary keys are used to ensure that data is *unique* - exists in one single location. NoSQL databases, on the other hand, due to their distributed nature, have no such limitation, and are therefore much more flexible and responsive to the changes one would experience in a connected system, for instance.

A. Direct Acyclic Graphs

IV. Results

V. Conclusion

Put the conclusions of the work here. The conclusion is like the abstract with an additional discussion of open points.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.