Nathan Mayer

Dr. Johnson

CS 200 Section 1

23 January 2023

Investigating Non-Relational Data Storage and Retrieval Systems

The modern world is inextricably dependent on technological systems capable of storing the vast amounts of data created and utilized by business, scientific research, medicine, and innumerable other fields. While many of the most widely utilized database management solutions employed to solve this challenge are relational in nature, relational databases are by no means the only type of systems being developed and implemented. This investigation will focus on two different overarching classes among these non-relational databases: "NoSQL" databases in general and, more specifically, graph databases.

The first of these two classes of non-relational database systems, NoSQL databases, have a number of interesting properties and use cases. Before delving into some of these properties and applications however, it would seem helpful to explain what exactly a NoSQL database is. A NoSQL database is a rather broad term used to describe any type of database that is structured using an alternate scheme when compared to more conventional relational databases, which use a series of connected tables, and have become prevalent since their conception by E.F. Codd in 1970. The NoSQL name itself refers to the fact that these databases were initially constructed so as not to use SQL, the standardized Structured Query Language that relational databases made use of in order to pose queries and interact with the database. However, since their inception, many NoSQL projects have ultimately come to add support for SQL usage due to its overall prevalence and popularity in the industry. Consequently, the NoSQL name is now said to stand for, "Not Only SQL" (Mullins, Vaughn, and Beal).

NoSQL projects were devised in order to address what were perceived as intrinsic and significant weaknesses in existing relational databases. These problems which NoSQL databases aimed to address can generally be summarized under the overarching theme that SQL based relational databases provided a one size fits all solution to database management that came along with disadvantages in terms of the associated hardware requirements, database performance, and overall adaptability (Pagan et. al.). NoSQL databases were designed to adapt their feature

sets more directly to the specific needs of a particular application or data repository rather than creating a ubiquitous standard. SQL based relational databases often were comparatively slow when it came to parsing through the entirety of the database, required expensive hardware to setup and manage, and depended on well-defined schema in order to operate (and thus could not easily evolve with the changing nature of the data that they stored) (Stauch). Relational databases were generally less than optimal for managing data that fell outside of strictly defined categories or was not easily organized in homogenous categories. Each time that new data had to be categorized or the relationship between tables of data had to be revised, the schema had to likewise be adjusted (a time and resource intensive prospect that became exponentially more problematic the more dynamic the nature of the data being dealt with). Relational databases were also not optimized for being broken up across more than one machine and could become slow as these additional storage locations strained their ability to maintain their well-defined internal data organization and access data across these various locations. For database needs that became so large that databases, by necessity, were required to be spread across more than one storage location, this became a significant concern (Lutkevich and Biscobing). All of these weaknesses were particularly highlighted as databases became increasingly integrated into web-based applications. In such web applications, the speed at which data could be accessed, the ease at which data could be spread across multiple locations, the relative flexibility to manage loosely correlated and vast quantities of unstructured data, and the ability of databases to be run on multiple comparatively lower-end servers, as opposed to single resource-heavy servers, became chief concerns (Stauch). Subsequently, new database management systems needed to arise that were able to be more finely tuned to the more specific demands of a particular application. NoSQL projects each attempted to meet these unique challenges in a variety of different ways.

Where relational databases store all data in a series of connected tables with a well-defined schema that dictates how this data is categorized with each table and how these tables interlink to each other, NoSQL databases may use any number of alternate data organization and storage methodologies (Lutkevich and Biscobing). Under the very broad umbrella of NoSQL database management systems, these alternate data organization and storage methodologies typically fall under one of four primary categories. These categories include, but are by no means limited to, graph databases, document databases, key-value store databases, and wide-column store databases (Mulllins, Vaughn, and Beal).

Graph databases, as stated, will be examined and considered as their own category of database management system later on in this investigation. Therefore, returning to the remaining three clades of NoSQL database management systems, document databases, as the name implies and as are represented by products like MongoDB are made up of base units known as documents. Rather than the tables used by relational databases, which have a schema that defines the columns and rows of each table, setting up the fields for each attribute before-hand, document databases store all of the data pertaining to a single database entry in a document. Documents can be grouped together into units known as collections. However, unlike in a relational database, each of the documents in a collection does not need to have a symmetrical layout in terms of fields when compared to the other documents in a collection, the schema can dynamically be changed on a document-to-document basis. Document databases are typically structured in a way to be similar to existing data storage paradigms common in object-oriented programming, in order to be more familiar and user friendly for database administrators (Mulllins, Vaughn, and Beal) (“What is a Document Database”).

Key-value store databases, as represented by Amazon’s DynamoDB, are structurally very similar to relational databases, but omit some key features used by relational databases in order to attempt to create a database management system that is faster and more resource efficient. Like relational databases, key-value store databases organize data into tables made up of columns of entries, known as items. Each of these items have attached data known as attributes (the same concept and term used by relational databases). Every item is given a primary key value designating where in the database it is stored (once again, this is identical to relational databases). However, key value databases go further to include composite primary keys, which not only designate where in a database a particular item exists, but also tags this data with a sort key that specifies how this item can be searched for with respect to other related items that are stored alongside it on the database. The sort key provides another way to further group and easily search through data. Sort keys are further augmented with secondary indexes, which allow the database designer to specify additional methodologies to sort through data that differ from the identifiers and methods described in the primary key. This allows for more precise and optimized search routines that can more efficiently sort through data in the database in the most efficient way needed to accomplish a particular data retrieval need. However, unlike relational databases, key-value stores omit the ability to join related tables. In a relational database, the database management software is able to merge and connect two tables that have the same columns,

meaning that this data does not have to be stored multiple times. This process is however, very resource intensive and requires that the columns of data be symmetrical across each joined table (DeBrie.)

Wide column store databases are represented by products like ScyllaDB. Like key value store databases, wide column store databases are structurally very similar to relational databases. They are also organized around tables with data placed into columns and rows. However, where wide column store databases diverge from traditional relational databases is in their ability to dynamically change the properties of table columns, including properties like their name and formatting parameters as needed, whether that be across different tables or the same table. Additionally, wide column store databases organize data such that it is stored on and retrieved from the disk by column. All of this is done in order to minimize the latency associated with database queries and maximize the database's ability to be organize data cohesively across more than one machine ("Wide-column Database").

Now, with this understanding of both the shortcomings of relational databases that NoSQL databases aimed to address as well as several different methods by which these principles were applied in order to produce database management systems not structured around the relational model, it would seem relevant to discuss what tangible benefits and drawbacks exist when it comes to actually using these NoSQL systems for database creation and administration.

In terms of advantages, NoSQL systems are typically better able to handle diverse data coming from multiple different sources in multiple different formats. The data doesn't have to be translated to a format that works well with the relational model and the schema does not typically have to be rewritten to accommodate these new types of data. NoSQL systems are typically more functionally compatible with web and cloud-based applications and are more amendable to being spread across more than one physical storage location. NoSQL is typically far more customizable for the exact data storage and retrieval needs of an application. The data for NoSQL databases also does not typically have to be normalized. Additionally, NoSQL databases tend to scale fairly well, typically handling latency with an expanding database size better than relational databases (Mullins, Vaughan, Beal) ("NoSQL Databases: Advantages and Disadvantages"). However, NoSQL systems are also not without disadvantages. Unlike with relational databases, NoSQL databases are not unified. Each project tends to use its own proprietary schema, and API/querying format. This makes them less interoperable and requires additional time to be

spend learning the features and structure of each specific NoSQL database management system. NoSQL projects also do not necessarily have the same universal standards pertaining to data integrity that are common across relational databases, and as such, can sometimes be more prone to data loss or corruption (Mullins, Vaughan, Beal) ("NoSQL Databases: Advantages and Disadvantages"). Now that NoSQL databases have been discussed in general, they can be examined more specifically for the case of the final type of NoSQL database, the graph database.

As mentioned earlier, graph databases represent a specific, targeted subset of the broader range of NoSQL databases. Where relational databases model data using tables made up of columns and rows, graph databases model data in accordance with graph theory, where each entity is an individual point, called a node. These nodes are able to be linked to each other according to their relationships, called edges. This is often represented pictorially with a series of points connected by lines to form a network (Rinson, Webber, and Eifrem) (Chiou et. al.) ("Graph Databases: How They Work, When to Use Them & the Advantages They Offer"). These systems were fundamentally created in order to provide more versatility in defining relationships between data when compared to traditional relational databases. Relational databases operate using well-defined and overarching schema that are difficult to alter as the needs of the database or nature of the data evolve. Graph databases were created in order to make it easier to update how parts of the database relate to each other over time ("Graph Databases: An Overview"). Some of the specific use cases of graph databases/problems graph databases attempt to solve include "social networks, fraud detection, asset management, recommendation and personalization, and …data mining," ("Graph Databases: An Overview"). Graph databases broadly use a framework that is built upon either Resource Description Framework graphs (which store divide the information contained within the graph into a object, predicate, subject paradigm) and property graphs (which detail the relationship between individual nodes of a graph) but can also be subdivided into many other graph types and applications, including social, intent, consumption, interest, mobile, property, and knowledge graphs ("Graph Databases: An Overview") ("Graph Databases: How They Work, When to Use Them & the Advantages They Offer").

While such descriptions of the general operating principles behind graph databases is useful in understanding them on a conceptual and historical level, it is far more helpful to look at how they have been more specifically implemented as actual database management system products. For instance, Neo4J is the most widely used graph database management system. Some of its unique features include a standardized query language and processor, known as Cypher, functioning very much akin to SQL for relational databases with many of the same

interoperability and ease of use advantages that come associated with a standardized query language. Neo4J also includes references to each connected node in the storage of each individual node of data, meaning that rather than the resource intensive master index of a relational database, Neo4J is able to store a lightweight indexing system that is fundamentally built into every data entry. This is said to significantly increase the speed at which data can be accessed and added to Neo4J databases, especially as the size of the database expands. Neo4J is designed to store data from the ground up in a graph optimized format, heavily using RAM over disk storage in order to minimize the number of read/writes being performed on the disk and subsequently optimize how quickly data can be accessed when requested. However, Neo4J inherits some of the disadvantages of relational databases in that it was not originally designed for use in databases that spanned multiple physical storage locations, and optimized support for such use cases is still being developed (Guegan).

Amazon's graph database platform Neptune also has a unique set of features and functions. Neptune is designed to support both of the primary graph database types cited earlier, having support for both property graphs and resource description framework graph structures. Each of these different types of graph databases have their own API. Property graph-based Neptune databases are accessed using the Gremlin API, similar in function to both SQL and the aforementioned Cypher query language discussed above for Neo4J. Gremlin has the unique benefit of being specifically designed for easy integration into end user applications, having been tailored for the purpose of facilitating easier user interactions with the database. Resource description framework-based Neptune databases however, use a query language known as SPARQL, which is designed to more easily import datasets from popular platforms as well as to export data from the database to other applications. Neptune is primarily designed to represent and store rather than process graph database data. It is designed to be highly redundant with a focus on data integrity (with data being spread and mirrored across units known as availability zones) as well as to support some of the largest graph sizes available (Anadiotis).

Now, considering both the general operating principles and specific implementations of graph database management systems, the question remains as to what the overall advantages and disadvantages of these types of NoSQL databases are, especially relative to relational databases. In terms of advantages, graph databases are able to model the same kinds of data as relational databases, being well suited for data that is highly dependent on interconnected items, but with significantly faster access times for that data regardless of the overall size of the database. Graph databases are more flexible than relational databases as well. Whereas typically data must be

translated or adapted for a relational database and then remain as consistent in structure as possible so as to avoid extensive schema rewriting, graph databases can be restructured comparatively more easily and adapted to changing data relationships over time. Graph databases are also well optimized for emergent technological fields like machine learning, as their node structure can easily be used to train AI models that not only are able to study and identify patterns in the relationships between graph nodes, but also learn to extend these relationships to new data. Graph databases also have some notable disadvantages. They are only faster and more resource efficient when storing data as compared to a relational database when that data is primarily described in terms of its relationships and connections. For data that exists as a straightforward list, relational databases are often simpler and more efficient. Additionally, while graph database systems are often faster in executing certain queries when compared to relational databases, they are also not able to respond to as robust and multileveled of queries as relational databases, having limits on how queried data can be sorted. Additionally, graph databases are often slow when it comes to having to parse through the entirety of the database. They work most optimally when searching for connected nodes from a given start node with a specific direction or relationship in mind. Furthermore, graph databases are slow and inefficient comparatively speaking when dealing with data that models transactions.

Works Cited

Alex DeBrie. *DynamoDB, Explained.*, https://www.dynamodbguide.com/the-dynamo-paper.

Anadiotis, George. "AWS Neptune Going Ga: The Good, the Bad, and the Ugly for Graph Database Users and Vendors." *ZDNET*, Red Ventures, 31 May 2018, https://www.zdnet.com/article/aws-neptune-going-ga-the-good-the-bad-and-the-ugly-for-graph-database-users-and-vendors/.

Chiou, Lawrence, et al. "Graph Theory." *Brilliant Math & Science Wiki*, https://brilliant.org/wiki/graph-theory/.

Foote, Keith D. "Graph Databases: An Overview." *DATAVERSITY*, 14 Apr. 2022, https://www.dataversity.net/graph-databases-an-overview/.

Foote, Keith D. "NoSQL Databases: Advantages and Disadvantages." *DATAVERSITY*, Dataversity Digital LLC, 17 Nov. 2022, https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/#.

"Graph Databases: How They Work, When to Use Them & the Advantages They Offer." *InfluxData*, InfluxData LLC, 28 Dec. 2022, https://www.influxdata.com/graph-database/#examples.

Guegan, Benjamin. "Neo4j Performance Architecture Explained & 6 Tuning Tips." *Graphable*, Graphable Inc., 30 Sept. 2021, https://www.graphable.ai/blog/neo4j-performance/.

Lutkevich, Ben, and Jacqueline Biscobing. "What Is a Relational Database?" *Data Management*, TechTarget, 24 June 2021, https://www.techtarget.com/searchdatamanagement/definition/relational-database.

Mullins, Craig S., et al. "What Is Nosql and How Do Nosql Databases Work?" *Data Management*, TechTarget, 8 Apr. 2021, https://www.techtarget.com/searchdatamanagement/definition/NoSQL-Not-Only-SQL.

Pagán, Javier Espinazo, et al. "A Repository for Scalable Model Management." *Software & Systems Modeling*, vol. 14, no. 1, 2013, pp. 219–239., https://doi.org/10.1007/s10270-013-0326-8.

Robinson, Ian, et al. *Graph Databases*. O'Reilly, 2015.

"What Is a Document Database?" *MongoDB*, MongoDB Inc., https://www.mongodb.com/document-databases.

"Wide-Column Database." *ScyllaDB*, 15 June 2022, https://www.scylladb.com/glossary/wide-column-database/.