NoSQL Database Architectures

COM644 Full Stack Web Development

Assignment 1

Dylan Keys

B00664777

Table of Contents

What is NoSQL? 2

Advantages of NoSQL over SQL 2

Disadvantages of NoSQL over SQL 3

Scalability 3

Cloud deployment 3

NoSQL models 4

Performance considerations 5

Example schemas and applications 6

References 7

# What is NoSQL?

NoSQL is the term used to describe non-relational databases. NoSQL databases are dynamic in their approach, in that they do not require a defined schema to get going. There are four key data models used by NoSQL databases: Document, Graph, Key-value databases and Wide column stores.

# Advantages of NoSQL over SQL

## Schema-less

SQL databases require a schema to be setup before the database can be used to input data. The schema will usually contain tables and their relationships, keys (primary, foreign etc). This must all be designed and decided ahead of any implementation. Changes can still be made down the line but can result in complications in making or after the change.

This is contrasted by the NoSQL approach, which is that data can be added anywhere, at any time. No schema design is needed ahead of implementation.

## Denormalisation

Is a technique that involves storing data everywhere it is required in the database, even if it is repetitive. This enables faster read operations.

The opposite, **normalisation** is the technique usually employed on SQL databases; where (for example) IDs are used to reference data in a table, rather than repeating this data in each place it is used. Therefore reducing data redundancy.

This is often used in NoSQL architectures, especially in document stores. Application’s that will benefit from read-optimisations can use this method to achieve faster loading of content/pages.

## Scalability and Performance

Scalability becomes quite difficult when using SQL database systems. Especially when thinking of scaling out to multiple servers to distribute the load. This can be partly attributed to requiring the performance and costly setup to synchronise data.

With NoSQL models, there is a focus on scalability more often than not. For example MongoDB has support for sharding partitioning and replication that allows for scaling out and enabling almost unlimited growth with a higher throughput and lower latency than standard SQL. (MongoDB, 2018).

## Models

NoSQL provides the developer with a number of options in terms of the data models. Much more so than SQL. These models (as described below) will allow for the development to be more tailored to suit the requirements of the application.

## Cost

Compared to SQL databases, the level of setup/infrastructure needed to support the architecture is small. This decreases the obscene costs that can sometimes be associated with SQL database hosting, setup and maintenance.

## Open Source

The open-source nature of NoSQL models makes the prospect of them much more enticing, especially to small businesses. Which saves the monetary and other hassles of licencing.

# Disadvantages of NoSQL over SQL

## Queries

NoSQL models loose some of the most powerful SQL queries, such as JOIN. In NoSQL models if normalised, data from multiple collections will need to be retrieved singularly and brought together in the application logic. This is one of emphasising points for denormalisation. Whereby one collection can be retrieved with all relevant data. This obviously will help speed up reads, compared to reading from multiple collections and having to connect within the code.

## CRUD Syntax

SQL is a well-documented, powerful declarative language that is an industry standard. NoSQL databases often use JavaScript-style queries with JSON-like arguments. The simple operations are easy, however things get more complex when working with nested JSON within complex queries.

## Reporting

There are a lack of reporting tools available for NoSQL databases, for analysis and performance testing. This is in contrast to SQL databases, specifically the market leader, MySQL, which contains a wide variety of reporting tools.

## Community

NoSQL is a rapid emerging technology but has not got the same amount of community backing that SQL has. When compared to the likes of MySQL which has a massive community of massively experienced users.

## Standardisation

As highlighted by Meijer and Bierman (Microsoft researchers), NoSQL needs standardisation in the form of a query language to grow (Meijer and Bierman, 2011).

# Scalability

Most NoSQL databases set out from the offset to be scalable. A lot of them have built-in scalability features, such as MongoDB which has built-in support for replication and sharding in order to support scalability.

The ability to scale horizontally on commodity hardware instead of singularly large servers make scaling simple, especially when compared with SQL databases.

# Cloud deployment

NoSQL models are exceptionally well suited to cloud-based platforms. They often are managed by third party vendors (such as AWS) who charge a subscription fee based on performance and scale. AWS are one of the largest vendors that offer cloud platform services on the web.

## Fast Deployment

The ability to quickly deploy code to a managed platform is one that entices a lot of businesses, especially those which do not have an in-house infrastructure team, that could handle the technicalities of setting up and managing servers.

## Ease of scaling

If your database was hosted on managed servers, the scaling process would be a tedious and lengthy one for the team involved. There are many techniques and methods that could be employed in order to achieve multi-server scalability, however when this is managed by a cloud-platform service provider it often takes the click of a button to scale up and down.

## Security

Another benefit of using a cloud-platform service provider is the security on offer. This is another vast area that would require much work and plenty of trial and error if managed in-house. With an established provider you know they are well-patched and secure from external threats. The new meltdown and spectre vulnerabilities are just one to note, if managed in-house it would be a mad dash to get patches applied due to the severity of the threat, with an established provided such as AWS you know they have the best working on alleviating any threat.

## Disaster Recovery

If the worst came to the worst and data was lost the in-house cost of setting up data recovery (DR) measures would be high. With cloud service providers this is an essential part of the service. If anything were to happen a roll-back would occur to the latest daily or hourly snapshot, depending on what is on offer.

## Support

Most providers will offer 24-7-365 technical support. So that if an issue were to occur, they would be on hand to assist. This takes some of the internal pressure off.

# NoSQL models

## Document Databases

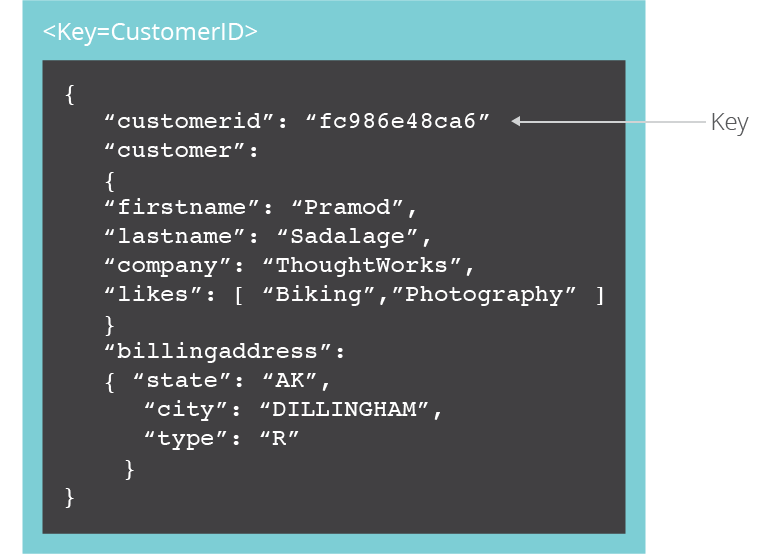
Document databases pair pieces of data with a key to form up a “document”. These documents are often in JSON, XML or BSON format. The keys are used to describe the contained data, and is presented in a hierarchical layout. Document databases, especially those in JSON, work well for developers that are looking to serialise and load objects containing the relevant properties and data (Amazon, no date a), especially when using JavaScript code. Examples of document databases are MongoDB, CouchDB and MarkLogic. A depiction of a document is shown in figure 1 (Sadalage, 2014).

Figure 1.

Document databases can be compared with key-value stores as they are similar in approach. Key-value stores are much simpler option. However, document databases offer much more flexibility to the developer. They look to offer a much higher level of organisation with data broke up into collections with much more in the way of querying the database. This is suited to applications that have a more complex dataset, that can make use of the collection organisation and queries.

## Columnar Databases

Columnar databases can be used in situations where query performance is key, as it reduces the overall disk I/O requirements as well as the amount of data that is loaded from the disk (Amazon, 2012).

## Graph Stores

Graph stores/databases are best suited to data that is highly connected. In this model the pieces of data are connected together with other related components via vertices and edges that may also have properties attached to them. This model’s best use cases are in areas with a lot of connected data, such as social applications, recommendation engines, fraud detection, knowledge graphs, life sciences and IT/network (Amazon, no date b). A depiction of a graph database can be seen in figure 2 (Sadalage, 2014).

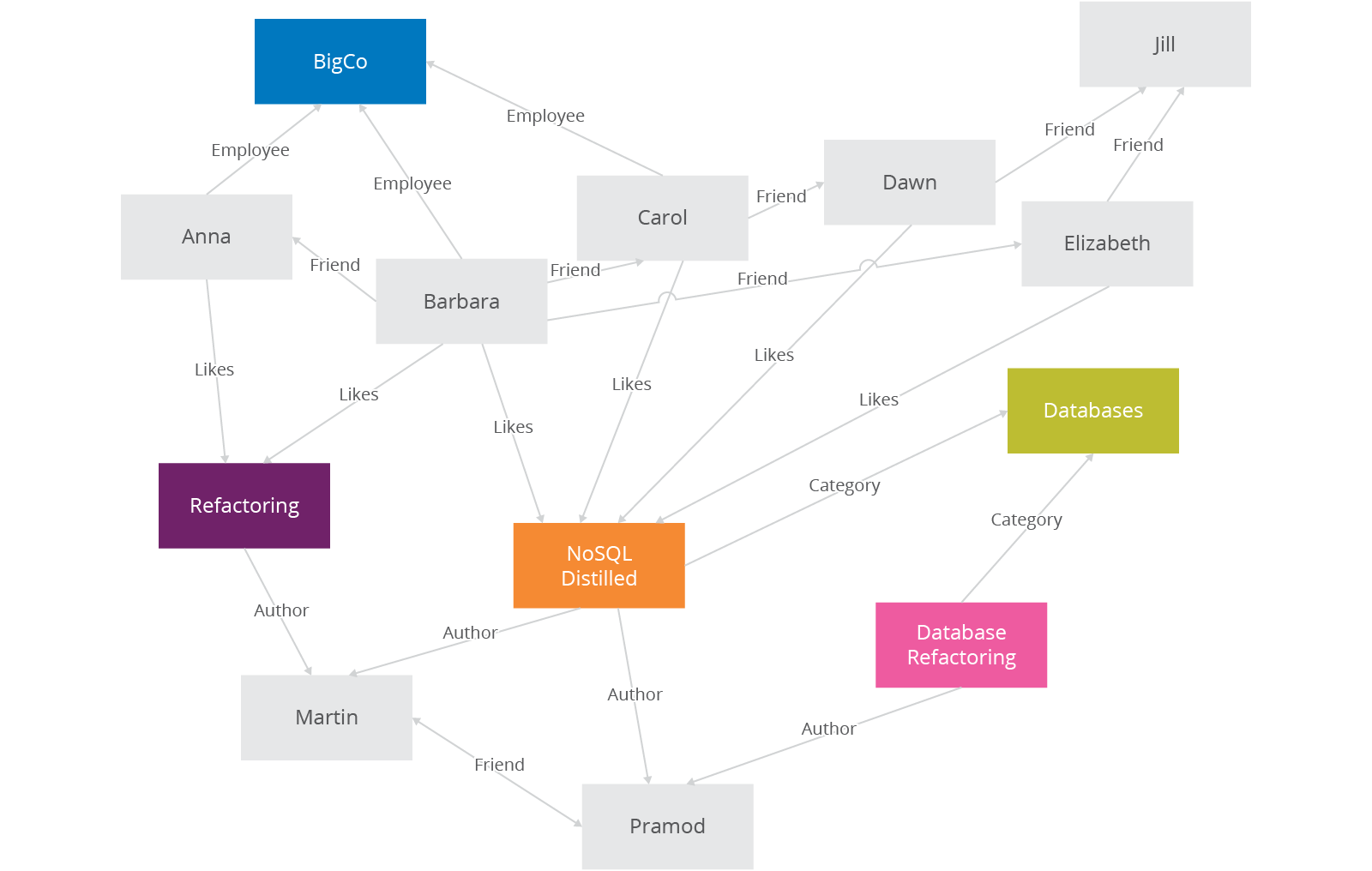


Figure 2

## Key-value Stores

In key-value stores is one of the most simplistic models. Data is stored in key-value pairs, where each key represents each value in the collection. Compared to others, it is much less complex, especially in terms of API use. Whereby the only operations a client can request are the value for a given key, to delete the key or add a value to a key. Values are stored as a blob, with no schema required. Therefore, data can be in any format; it is up to the application code to understand what kind of data is stored.

In comparison to **document databases**, key-value stores are the simplistic option. Both could be described with document databases having more functionality. However, this doesn’t necessarily mean that they are better. Like in most model selection scenarios it comes down to the application needs. Key-value stores are the best option when simple searching is what is required. If the needs are to retrieve data based on key or ID value, then there is no need for a heavier model with complex queries that could effect performance.

# Performance considerations

## CAP Theorem

Defined by Eric Brewer, CAP theorem states that at most in a distributed system you can only have two of the following (Brewer, 2012):

* Consistency (C) equivalent to having a single up-to-date copy of the data
* High availability (A) of that data (for updates)
* Tolerance to network partitions (P)

A number of NoSQL databases will allow the developer to have more control in deciding on which components to refine, i.e. in which area to be strong and which to sacrifice.

# Example schemas and applications

## MongoDB

MongoDB products are revolved around document databases, which use collections to maintain organisation. Highly integratable with JavaScript code.

Main features:

* Indexing – the use of primary and secondary keys, and automatic key generation
* Replication - replica sets contain 2+ copies of stored data
* Load balancing – MongoDB can be used over multiple servers to balance the load or for replication purposes in active/passive form
* File storage – MongoDB can be used for storing files across multiple machines with load balancing and data replication
* Aggregation – with the use of MapReduce batch processing of data can be achieved, along with aggregation operations which will produce a similar effect to SQL GROUP BY clause.
* JavaScript execution – JavaScript can be used in queries and sent direct to the database in order to be executed
* Capped collections – MongoDB allows the creation of set size collections

## Redis

Redis is an in-memory key-value store, used as a database, cache and message broker (Redis, 2018). It is one of the most popular key-value stores and used by large companies such as Twitter. One of the main uses of Redis is to store session cache data, this works well with the Redis architecture. Redis can achieve a high level of scalability by using clustering, which it supports up to 1,000 nodes, within this technology is the allowance for some nodes to fail and operations to continue. Also, master-slave replication is available for extra data safety.

# References

Amazon (2012) *Columnar Storage*, *AWS*. Available at: https://docs.aws.amazon.com/redshift/latest/dg/c\_columnar\_storage\_disk\_mem\_mgmnt.html (Accessed: 20 February 2018).

Amazon (no date a) *What is a Document Database?*, *AWS*. Available at: https://aws.amazon.com/nosql/document/ (Accessed: 20 February 2018).

Amazon (no date b) *What is a Graph Database?*, *AWS*. Available at: https://aws.amazon.com/nosql/graph/ (Accessed: 20 February 2018).

Brewer, E. (2012) *CAP Twelve Years Later: How the &quot;Rules&quot; Have Changed*, *InfoQ*. Available at: https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed (Accessed: 28 February 2018).

Meijer, E. and Bierman, G. (2011) ‘A co-relational model of data for large shared data banks’, *Communications of the ACM*, 54(4), p. 49. doi: 10.1145/1924421.1924436.

MongoDB (2018) *MongoDB Architecture*, *MongoDB*. Available at: https://www.mongodb.com/mongodb-architecture (Accessed: 1 March 2018).

Redis (2018) *Introduction to Redis*, *Redis*.

Sadalage, P. (2014) *NoSQL Databases: An Overview*, *ThoughtWorks*. Available at: https://www.thoughtworks.com/insights/blog/nosql-databases-overview (Accessed: 20 February 2018).