# Evaluation of Relational and Non-Relational Databases

Gregory Walsh
29785685
gw2g17@soton.ac.uk

## ABSTRACT

In this essay, a comparison of relational and non-relational databases is given along with a description of the benefits and costs of different non-relational databases. A hypothetical scenario involving selecting an appropriate database for the purpose of analysing the Enron email dataset is also discussed, and it was concluded that the choice depends on the technical skills of the analyst and her/his team, and the type of analytical task to be executed.

## 1 INTRODUCTION

Over the next three years, the volume of data being generated and consumed is expected to double every year, and the structure of that data is becoming ever more complex and varied. The cost of on-line data storage is dropping too and by 2020 the unit price is expected to be one tenth of what it was in 2012. These changes in demand and costs are have caused a major shift in the way organisations approach data storage [1].

One of the main factors driving the increase in data generation is the increase in unstructured textual and image data, which is highly heterogeneous and lacking obvious organisational structure. The systems which store structured data, known as "relational databases" (RDBs), are still common today, but they are not well suited to handling the volume and heterogeneity of unstructured data. To cope with these new demands, new technologies have been developed. These new databases known as "NoSQL" or "non-relational" databases (non-RDBs) have enabled the collection and analysis of data that would have otherwise been impossible. [2].

## 2 DIFFERENCES BETWEEN RELATIONAL AND NON-RELATIONAL DATABASES

### 2.1 How is The Data Stored?

In the late 1960s, Edgar Codd defined a new model for data storage and retrieval which would become the blueprint to the RDB. At a time when storage of data was at a premium, he was dissatisfied with the inefficiency at which the contemporary data bank technologies stored structured data. The linked-list storage method, which these data banks used, had two main problems. Firstly, records with only a subset of the properties allowed by the data bank would take up just as much space as complete records. Secondly, information which was common to many rows had to be duplicated for every row in the data bank.

The model he created to solve these and other problems was a set-based one, describing three key concepts: relations, attributes, and tuples (we now know them as tables, columns and rows respectively), and the relationships between them. In this model, the data is normalised, which is to say the relations and attributes are organised in such a way as to minimise the amount of repeated information in the database [3]. From this mathematical model, the RDB and the declarative structured query language (SQL) were developed.

In contrast with the normalised storage structures on which RDBs operate, non-RDBs store data in a range of non-normalised ways. Common to all is that each item of data has their own unique identifier. Because the information is non-normalised, there may be significant redundancy of data, however, due to the massive decrease in disk space costs in recent times, this inefficiency is no longer as much of an issue.

### 2.2 What Kinds of Data Types Do They Suit?

To maximise the benefits of storing data in a RDB by taking advantage of normalisation, the records comprising the data must be highly structured, which is to say that they share with one another an almost identical composition of properties, and that the values of those properties are also similar.

Conversely, non-RDBs are capable of storing both structured and unstructured data such as text, images and video. When high read/write performance of unstructured data is required, non-RDBs, when deployed on a distributed system (such as a cluster), are particularly suitable since nodes can act independently to both serve and ingest data. [4]

### 2.3 What Guarantees Do They Make?

Following Codd's work, Reuter and Härder identified a set of four guarantees, collectively known as "ACID", which are desirable to have when working with RDBs. The first is atomicity, meaning all or none of a transaction will be committed. Next, consistency states that database constraints will never be violated. Isolation specifies that if two or more pending transactions touch a shared record, the resulting state of the database and any results will be as if the transactions had been executed in full one after another. Finally, durability states that after a transaction has been successfully committed, the results of that commit are persistent and resilient to system crashes. [5]

In contrast to the ACID guarantees of RDBs, the backronym BASE was coined for non-RDBs. It includes the following: a "basically available" guarantee (i.e. available according to the

"CAP" definition - see section 2.4); a statement that the database is "soft-state" and therefore liable to change between queries; and a weak "eventually consistency" guarantee (according to the CAP definition of consistency) [6].

## 2.4 How Is Performance Scaled?

The performance of both RDBs and non-RDBs can be increased by running them on more powerful hardware, an approach known as "vertical scaling", however, this is often expensive [7] and has its limits since a single computer can only ever be so powerful.

As an alternative, computer scientists began thinking about horizontal scaling, whereby the database processes and data could be distributed across multiple affordable commodity servers. However, this approach was found to have a significant limitation. In the late 1990s, Eric Brewer developed the CAP conjecture (which then later became a theorem) which describes a set of three desirable guarantees, of which at most two can be fulfilled in the event of a node failure in a distributed (horizontally scaled) system. These are: consistency (defined differently from the database level consistency of ACID), meaning regardless of which node services the request, the result will be the same; availability, meaning reads always get a response, but the record returned may not be the most recent version; and partition tolerance, meaning finite messaging failures between nodes do not affect the behaviour of the system when viewed from the outside [8]. For most use-cases, recovery from hardware failure is an essential operating requirement, therefore most distributed RDBs and non-RDBs are designed to have partition tolerance.

Since historically RDBs have provided atomicity and consistency as part of ACID, a pair of essential guarantees for many use-cases (for example storing and manipulating financial records), distributed RDB systems are designed with consistency in mind, by accepting that partition failure may cause availability problems.

In contrast, non-RDBs sacrifice consistency for availability, since they are often used in cases where recent but not current information is acceptable (for example showing current results of an internet poll) [9]. By sacrificing strict consistency, dividing the work among many nodes, and allowing redundant data, distributed non-RDBs can cope with far higher incoming and outgoing rates of data than equally sized distributed RDB systems, which are unable to equally divide the processing among all their nodes.

## 2.5 How Is the Data Queried?

RDBs offer a very rich set of operations for querying data, supporting almost any combination of row selection, aggregation, set, and join operations through the use of SQL. Since data is often stored in a normalised format, supporting such a broad variety of operations is essential to extracting several features of data objects, since an object's attributes may be distributed among several tables. The query flexibility of RDBs also makes it an attractive choice to analysts without software development skills, since SQL is declarative and makes it possible to perform complex queries without having to program the implementation of the extraction pipeline.

In contrast, non-RDBs are interfaced with a variety of different query languages. Though some non-RDBs support declarative SQL-like languages, often only a subset of the operations which can be performed by RDBs are available. As a result, post-processing of data using another tool may be required.

# 3 ADVANTAGES AND DISADVANTAGES OF DIFFERENT KINDS OF NON-RDBs

There are several kinds of non-RDBs. In this section, in general terms and in the context of the Enron data set, non-RDBs from several different categories (graph type, document store, key value store and wide column store) are discussed.

In October 2001, fraudulent accounting activities at Enron became public knowledge, ultimately causing the collapse of the company, and massive loses for investors. After the ensuing investigation, a huge collection of employee emails was made available to the public. No other email collection of this size exists in the public domain. Considering these facts, it is reasonable to assume that a person analysing the data set may have a general interest in the statistics of electronic communication within large organisations, or alternatively a particular interest in the events that led up to the scandal. It is from the first of these two perspectives, in addition to a general perspective, that the relative advantages and disadvantages of non-RDBs will be compared.

## 3.1 Graph Databases

In comparison to RDBs, each category of non-RDB addresses quite different operational requirements. However, of the four categories, the graph type databases could be considered as addressing the most specific need: analysing highly interconnected data. Graph databases like Neo4j store the records and their relationships between those records as nodes and edges respectively. These databases offer significantly higher performance on network type queries [10] since the database does not need to determine at runtime the relationships between documents. This improvement in performance enables analysts to more easily explore the relationships between nodes in a network and identify interesting features. For example, determining which twitter accounts are likely to be bots.

In the context of the Enron data set, if our hypothetical analyst was interested in understanding the relationships between different employees, perhaps as a way to identify people who were likely to be involved in, or instigators of illegal activity due to their associations with other individuals who have already been identified as complicit in fraudulent behavior, graph databases would provide the functionality they require. The visualisation tools provided by Neo4j would most likely also be of help when performing this kind of analysis.

A downside to graph databases is they are difficult to horizontally scale successfully, unlike most other non-RDBs [11]. However, progress is being made on this front and Neo4j now has much better sharding feature set that was not available previously.

### 3.2 Document Store Databases

Document store databases, such as MongoDB and CouchDB, are particularly suited to querying semi-structured data with hierarchical or nested properties. They provide a broad set of tools to access, manipulate and aggregate based on shared properties across documents, and so provide application developers with the flexibility to modify the structure of documents, whilst still maintaining the ability to do relatively complex searches and aggregations [12]. Document store databases like MongoDB even support some relational algebra operations, such as "lookup", which databases in other categories do not typically support.

At this point, it is important to mention that the Enron data is comprised entirely of emails, that all emails in the data set share a set of basic properties such as sender and send time, and that these common properties are organised in a hierarchical structure. However, there are some other fields which do not occur in every email, such as "Cc", so the data is semi-structured with a flexible schema. As an analyst interested in performing aggregations or searches on email data, for example to determine which period in the day sees the highest volume of email sent, the combination of schema flexibility and querying functionality of document store databases make them an attractive potential choice.

As with any distributed database which prefers availability over consistency, document store databases come with a sting in the tail for application developers, since they do not provide the atomicity and consistency guarantees on transactions that traditional RDBs do [13]. However, with MongoDB this is somewhat mitigated by the ability to perform atomic operations on individual records using functions such as "findAndModify". Furthermore, some document store databases have a single master node which distributes writes to the other nodes in the cluster, so if this master node fails then the database will be unable to write new information to disk. From the perspective of an analyst working with the Enron data set, which is a fixed size, and need only be loaded a single time into the database, neither of these issues are of much concern.

### 3.3 Key Value and Wide Column Store Databases

Key value databases, such as Riak and wide column store databases like Casandra are designed with high availability and dynamic scalability [Dynamo] in mind for both reads and writes. These categories of databases are particularly useful in situations which require very low latency, such as serving on-line advertising content. Wide column stores provide additional functionality over key value databases in the form of more granular access to the specific properties of records and high-performance aggregations over columns.

However, though databases like Casandra support aggregation, the usefulness of these databases to an analyst is diminished by the restricted set of possible operations when compared to other non-RDBs [12]. As such, they offer little in the way of useful features to analysts interested in extracting insights from semi-structured data like the Enron corpus, since extremely high availability carried no benefit for systems with a single user.

## 4 CONCLUSIONS

Determining which database will best facilitate the research of an analyst depends on the kind of analysis the analyst wishes to conduct, their technical abilities, and the level of database programming support their organisation can provide them with. With respect to the Enron data set, should the analyst hope to understand and explore the relationships between individual employees, Neo4j would be a reasonable choice, due to performance benefits of processing network data, the visual interactive interface, and its declarative query language "Cypher" which makes it more accessible to analysts with limited knowledge of programming.

Alternatively, if it is the analyst's goal to perform complex aggregations across the entire data set, and the analyst has a reasonable understanding of aggregation pipelines and a basic grasp of a programming language such as Python for the purposes of final data post-processing, either CouchDB or MongoDB would be acceptable choices, especially since CouchDB now supports a new query language based on the MongoDB syntax.

Finally, in the situation that the analyst has little programming experience other than SQL, and is well supported by colleagues with experience in extract-transform-load operations (a situation common in large organisations), since the dataset is of a fixed size and contains fairly well-structured data, there exists the option to normalise it and store it in a RDB. In the interests of read performance, a columnar storage RDB such as Vertica would be preferable.

## REFERENCES


[1] Gantz, J. and Reinsel, D., 2012. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze the future, 2007(2012), pp.1-16.

[2] Kaisler, S., Armour, F., Espinosa, J.A. and Money, W., 2013, January. Big data: Issues and challenges moving forward. In System Sciences (HICSS), 2013 46th Hawaii International Conference on (pp. 995-1004). IEEE.

[3] Codd, E.F., 1970. A relational model of data for large shared data banks. Communications of the ACM, 13(6), pp.377-387.

[4] Cattell, R., 2011. Scalable SQL and NoSQL data stores. Acm Sigmod Record, 39(4), pp.12-27.

[5] Haerder, T. and Reuter, A., 1983. Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR), 15(4), pp.287-317.

[6] Pritchett, D., 2008. Base: An acid alternative. Queue, 6(3), pp.48-55.

[7] Leavitt, N., 2010. Will NoSQL databases live up to their promise? Computer, 43(2).

[8] Gilbert, S. and Lynch, N., 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. Acm Sigact News, 33(2), pp.51-59.

[9] Brewer, E., 2012. CAP twelve years later: How the" rules" have changed. Computer, 45(2), pp.23-29.

[10] Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y. and Wilkins, D., 2010, April. A comparison of a graph database and a relational database: a data provenance perspective. In Proceedings of the 48th annual Southeast regional conference (p. 42). ACM.

[11] Nayak, A., Poriya, A. and Poojary, D., 2013. Type of NOSQL databases and its comparison with relational databases. International Journal of Applied Information Systems, 5(4), pp.16-19.

[12] Kaur, K. and Rani, R., 2013, October. Modeling and querying data in NoSQL databases. In Big Data, 2013 IEEE International Conference on (pp. 1-7). IEEE.

[13] Brewer, E., 2012. CAP twelve years later: How the "rules" have changed. Computer, 45(2), pp.23-29.


*I am aware of the requirements of good academic practice, and the potential penalties for any breaches.*