A Framework for Selecting NoSQL Databases: a NetFlow Use Case

by

Leon Albertus Rheeder

A Framework for Selecting NoSQL Databases: a NetFlow Use Case

by

Leon Albertus Rheeder

Dissertation

submitted in fulfilment
of the requirements
for the degree

Master of Information Technology

in the

Faculty of Engineering, the Built Environment and Information Technology

of the

Nelson Mandela University

Supervisor: Prof. Reinhardt A. Botha

March 2018

DECLARATION OF ORIGINALITY

I, Leon Albertus Rheeder, hereby declare that:

- The work in this dissertation is my own work.
- All sources used or referred to have been documented and recognised.
- This dissertation has not previously been submitted in full or partial fulfilment of the requirements for an equivalent or higher qualification and any other recognised educational institute.

Leon Albertus Rheeder

ACKNOWLEDGEMENTS

I would like to thank the following individuals for their unconditional encouragement, support and guidance:

Thank you to my Heavenly Father, for enabling me with the courage, strength and ability to undertake and complete this study.

Thank you to my supervisor, Professor Reinhardt Botha, whose time, direction and support taught me so much. I would also like to thank Professor Reinhardt Botha for his patience, enthusiasm and vast knowledge for guiding me throughout this process. Thank you, Professor.

Thank you to my parents and girlfriend who have been my motivation and support system throughout this entire process.

I would also like to thank the following benefactors for their financial assistance:

- The financial assistance of SANReN, CSIR towards this research is hereby acknowledged. Opinions expressed, and conclusions arrived at, are those of the authors, and not necessarily to be attributed to SANReN, CSIR. Thank you.

- The financial assistance of the Nelson Mandela University's Post Graduate Research Scholarship (PGRS) is also hereby acknowledged. Thank you.

ABSTRACT

Making decisions regarding technology is difficult for IT practitioners, especially when they lack formal guidance. Ad hoc decisions are prone to be influenced by biases. This research study specifically considered decisions regarding NoSQL. The primary objective of this study was to develop a framework that can assist IT practitioners with decisions regarding NoSQL technologies. An investigation into typical decision-making problems encountered when having to make technology-based decisions provided an understanding of the problem context. The application context was explored through a literature study of the four NoSQL families.

This study produces a framework to assist IT practitioners in making decisions regarding technology. The framework comprises two models. Firstly, a weighted decision model combines several constructs, thereby providing a general method of making decisions. Secondly, a 6-step process model that can be used to adapt the weighted decision-model to a specific type of technology and a specific use case is proposed.

The feasibility and utility of the proposed framework are demonstrated by applying the framework to a NetFlow use case. If NetFlow data is to be used for analytical decision-making, the data must be stored long-term. NoSQL databases have increased in popularity, especially in decision-making contexts. Therefore, NoSQL is a logical storage choice. However, which NoSQL family to use is not self-evident. Therefore, the decision-maker may require assistance to make the right decision.

To assist with this decision, the framework was adapted to be used in the NoSQL context. A set of criteria was developed to allow various NoSQL options to be uniformly compared. Furthermore, the four NoSQL families were graded based on this set of criteria. After adaptation, experts provided input regarding the requirements of the NetFlow use case. This resulted in the weighting of the criteria for this specific use case. Finally, a weighted score was calculated for each family. For the NetFlow use case, the model suggests that a document-based NoSQL database be used.

The framework ensures that all NoSQL technologies are systematically investigated, thereby reducing the effect of biases. Thus, the problem identified in this study is addressed. The proposed model can also serve as a foundation for future research.

TABLE OF CONTENTS

PART D - EPILOGUE

LIST OF TABLES

LIST OF FIGURES

PART A

CONTEXT

CHAPTER 1: INTRODUCTION

The *Oxford English Dictionary* (2017) defines a *decision* as "a conclusion or resolution reached after consideration". Individuals make decisions regarding all aspects of life and draw countless conclusions in a variety of contexts. Decision-making is a core process of daily life (Nooraie, 2012). Some decisions are made knowingly and others unknowingly (Kahneman & Tversky, 1984). Decisions can be high-risk, for example deciding to go to war, or low-risk, for example deciding to buy a loaf of bread (Kahneman & Tversky, 1984).

Some decisions are easy, while other decisions are difficult. Easy decisions do not require much effort from the individual. Difficult decisions require more effort from the individual because they require more information to be considered or because their outcomes are of higher importance (Kahneman & Tversky, 1984). As a result, the chance of making a wrong decision increases. There are two factors, namely decision-making biases and measurements, that can influence the decisions that individuals make. Behavioural economists have argued that all decision-makers are subject to biases. Analytical individuals could use measurements to inform decisions and fight these biases. However, the decision-maker's view on measurement would introduce further biases. If decision-makers are not cognisant of how measurements and biases may influence them, this could lead to incorrect decisions being made.

As mentioned before, decision-making is part of many aspects of life, ranging from mathematics and statistics, through economic and political science, to sociology and psychology (Kahneman & Tversky, 1984; Nooraie, 2012). In this research study, decision-making regarding technology is the specific domain of interest.

1.1 Technology decision-making

Decision-making in general is a broad concept to consider and discuss. This study will focus specifically on decision-making concerning technology. Making technology decisions requires an IT practitioner to consider several aspects that could be influential. Examples of such aspects include *information overload* (Speier, Valacich & Vessey, 1999), the *lack of information regarding a technology* (Cowan, 1991; Desouza, Jha, Papagari & Ye, 2006), and the *documented use cases of the technology* (Hoff, 2011). *Information overload* refers to the constantly expanding and growing plethora of information regarding technologies that must be taken into consideration when making technology decisions (O'Reilly, 1980). The volume of information can be too great to consider at once, thereby increasing the difficulty of making the technology decision and increasing the chances of making the wrong decision.

The *lack of information regarding a technology* refers to how unknown the information regarding a specific technology is (Cowan, 1991). Relatively new technologies, such as NoSQL (Not Only SQL), are not as common as relational databases. Therefore, individuals may not have much information regarding the technology (Leavitt, 2010). The uncommon nature of and lack of information regarding the technology increases the difficulty of making a technology decision.

Documented use cases of the technology refer to the common practices and requirements regarding a specific technology (Kulak & Guiney, 2012). Known use cases are well documented, which decreases the difficulty of making decisions (Jacobson, 2003). However, a use case is just a specific case and individuals may not know how to apply or customize it to specific requirements, for example the requirements to store NetFlow data within a NoSQL database. In this case, current use cases may not be particularly helpful, because NetFlow data is not commonly associated with NoSQL. An IT practitioner may not know how to decide which technology is best suited for this specific use case.

1.2 Problem area

The above-mentioned examples are based on real-world cases of decision-making. They illustrate the real-world problem that *decision-making is a difficult task to complete*. However, decision-making without a context is too broad and general to focus on. Therefore, this study uses a specific context that represents technology decision-making concerning NoSQL.

When the term NoSQL first appeared in 1998, it referred to a relational database system that did not employ SQL as a querying language (Strauch, Sites & Kriha, 2011; Strozzi, 2010). The term reappeared in 2009 in a conference set up by Jon Oskarsson that focused on non-relational database systems (Evans, 2009) and has increased in popularity ever since. NoSQL now refers to a type of database management system that is non-relational (Naheman & Wei, 2013) and was created to address certain limitations of relational databases. There are four categories of NoSQL databases: *key-value stores*, *column-family stores*, *graph stores*, and *document-based stores* (Aniceto, Xavier, Guimarães, Hondo, Holanda, Walter & Lifschitz, 2015).

The use cases for NoSQL databases are well documented. *Key-value stores* are well geared to handle use cases that include quick retrievals or updates, such as managing user profiles or managing web sessions (Moniruzzaman & Hossain, 2013). *Column-family stores* can store a variety of data types and involve large volumes of data. A common use case for *column-family stores* is storing and managing Facebook messages (Dimiduk, Khurana, Ryan & Stack, 2013). *Graph stores* focus on linked and relationship-heavy data. Therefore, common use cases are fraud detection and social networking (Hecht & Jablonski, 2011). *Document-based stores* involve large volumes of data. Common use cases include content management and event logging (Magnusson, 2013).

Storing NetFlow data in a NoSQL database is an uncommon use case. Its uncommon nature may lead to individuals not knowing how to handle this use case. If a use case is not well documented, the IT practitioner is required to find more information and decide which NoSQL family to choose. Therefore, the less common the knowledge regarding a technology is, the more wrong decisions are likely to occur. This leads to the *research problem* for this study.

1.3 Problem statement

IT practitioners do not have a systematic way to select the NoSQL family for non-arbitrary use cases.

1.4 Research objectives and questions

The primary research objective of this study is to create a framework to help IT practitioners with NoSQL decisions. To achieve this objective, three research sub-objectives need to be addressed.

Sub-objective 1 (SO1) : Enumerate typical decision-making problems when choosing between technologies.

Sub-objective 2 (SO2) : Identify a general model of decision-making.

Sub-objective 3 (SO3) : Create a process to tailor the approach to the NoSQL scenario.

1.5 Research approach

The problem addressed by this research study suggests an artifact in the form of a decision model to help IT practitioners with NoSQL decisions should be created. Therefore, the design of the study is influenced by design science research. This study employs the design science research framework of March and Smith (1995) with a focus on IT research. To facilitate the achievement of the study's goal, the research framework will provide direction for the research, discussion, and argumentation within this study. The framework views research outputs on four levels of abstraction, namely, *constructs*, *models*, *methods*, and *instantiation.*

The first research output, *constructs,* refers to "concepts form the vocabulary of a domain" (March & Smith, 1995, p. 256). Constructs are the basic terms and concepts used to describe an area, a situation, or a problem. The constructs can be formal or informal as long as they define the terms used to describe and think about tasks (March & Smith, 1995). Examples of formal constructs in the context of relational databases include rows, columns, and relationships. Examples of informal constructs include agreement, dissatisfaction, and participation.

The second research output, the *model*, refers to *"*a set of propositions or statements expressing relationships among constructs" (March & Smith, 1995, p.256). The model describes the relationships between the constructs to represent the situation or problem. To be a useful

representation, a model needs to capture the structure of the situation through the constructs and their relationships (March & Smith, 1995).

The third research output, the *method*, refers to "a set of steps (an algorithm or guideline) used to perform a task" (March & Smith, 1995, p. 257). The method is essentially based on the fundamental concepts (constructs) and the relationships between the concepts (model). The method takes various inputs from the model to create steps to perform tasks (March & Smith, 1995).

The last research output, *instantiation*, refers to "the realization of an artifact in its environment" (March & Smith, 1995, p. 258). Instantiations are used to provide a context for the operationalisation of the constructs, models, and methods. They "demonstrate the feasibility and effectiveness of the models and methods they contain" (March & Smith, 1995, p. 258). Thus, instantiation provides the first level of evaluation by showing that it is indeed feasible to construct the artifact and that the artifact is useful.

1.6 Research design and reporting

This study can be broken into four parts representing the four outputs of March and Smith's (1995) research framework. These can also be mapped to the objectives of this study. Figure 1.1 is a graphical representation of the relationship between the framework of March and Smith (1995) and this study's sub-objectives.

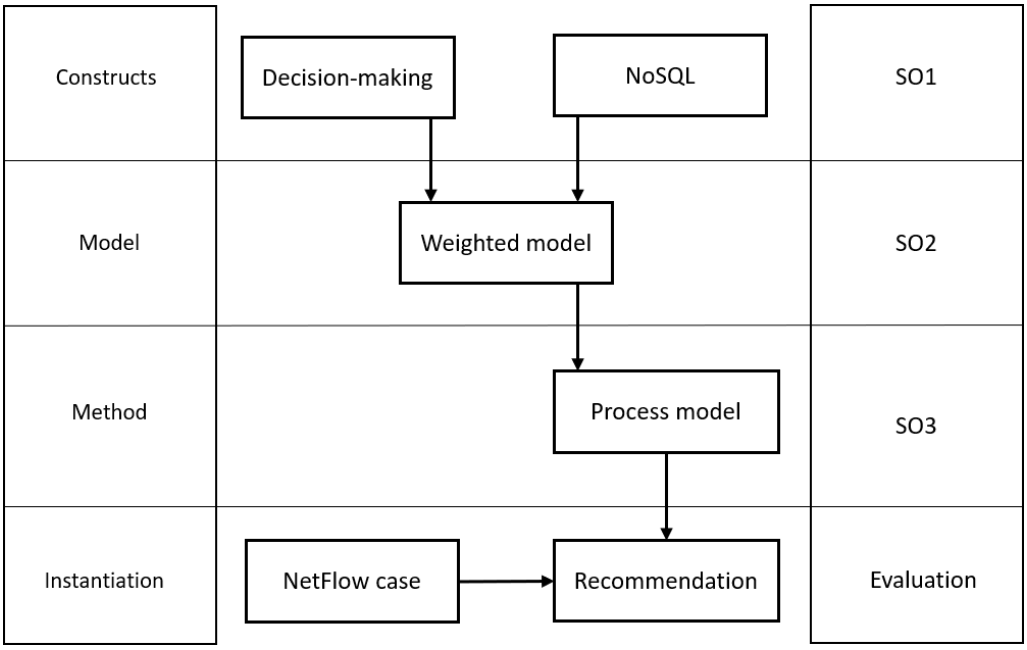

Figure 1.1: Mapping research objectives to March and Smith's (1995) design science framework.

The study places focus on the *build* research activity. Build refers to "the construction of the artifact, demonstrating that such an artifact *can* be constructed" (March & Smith, 1995, p. 258). By focussing

on building an artefact, feasibility can be demonstrated (March & Smith, 1995). Therefore, each of the four parts must be built to demonstrate the feasibility of the framework.

Part A deals with the *context* of this study. The main topics that need to be defined are decision-making and the problems that influence decision-making within a technology context. These problems include biases and the role of measurement. The technology context used within this study is that of the NoSQL families. The context of this study is described in Chapters 1, 2 and 3 and indicates the study's problem situation.

Part B deals with the construction of the framework. The framework is made up of constructs, a *decision model*, and a *process model*, and is discussed in Chapter 4. The *constructs* used in the study are a list of choices, a fixed set of criteria, the weights of criteria, grades, score calculation, and process steps. Firstly, the list of choices refers to the four NoSQL families. Secondly, the fixed set of criteria refers to the criteria used to compare the families uniformly. Thirdly, the weights of criteria refer to the importance of each of the criteria. Fourthly, the grades refer to the performance of each NoSQL family pertaining to the criteria. Fifthly, score calculation refers to calculating the final score of each NoSQL family. Lastly, the process steps refer to the steps followed to implement the framework. Sub-objective 1 (SO1) is met through a literature survey that defines the constructs used in this study.

The *decision model* within the framework deals with the relationships between the constructs. A basic overview of the decision model is discussed in Chapter 4. Sub-objective 2 (SO2) is met through a literature survey that identifies a framework and a mathematical expression to depict the framework within this study.

The *method* used to accomplish the task of the framework consists of a 6-step process. Step 1 is to investigate the technologies, and is presented in Chapter 3. Step 2 is to create a fixed set of criteria. The criteria discussion influences only NoSQL and is presented in Chapter 5. Step 3 is to grade the criteria in order to be able to compare the NoSQL technologies. Step 4 is to assign weight values to the criteria. Step 5 is to score each of the NoSQL technologies, and Step 6 is to provide a recommendation. Steps 3, 4, and 5 are discussed in Chapter 6, while Step 6 is discussed in Chapter 8. Sub-objective 3 (SO3) is met through argumentation and a literature survey that expands on the constructs within this study to tailor its approach to the NoSQL scenario.

Essentially, all the objectives of the study are met in Parts A and B. However, questions regarding the feasibility and usefulness the model may arise. Therefore, Part C demonstrates the feasibility and usability of the model.

Part C deals with *instantiation*, which refers to the use case within the study. The artefact is placed within a specific instance (use case) to verify and demonstrate its use. At the start of the verification, the NoSQL technologies are graded to ensure a uniform comparison can be made. This is described in Chapter 7. The specific instance (use case) for this study concerns NetFlow data, which has certain requirements that need to be considered. These requirements are reflected in weights that are assigned to the criteria in Chapter 8. Once the weights are assigned, the final score for each NoSQL technology can be calculated. Thereafter, a recommendation can be provided. This is also discussed in Chapter 8. The instantiation of the artefact in the last part of the study will verify that the model can be utilised.

Part D is the *epilogue* and contains only one chapter, Chapter 9, which deals with a reflection on what was done and how the objectives of this study were met. Thereafter, limitations and future work are mentioned to conclude this study.

All the above-mentioned points are brought together through argumentation to achieve the main research objective of creating a framework to help IT practitioners with NoSQL decisions. To indicate whether the objective was met, a NetFlow use case is employed to demonstrate the feasibility and utility of the model.

1.7 Conclusion

Making the right decisions regarding technologies is important and difficult. As mentioned above, there are a variety of elements that could influence such a decision, including *information overload*, the *lack of information regarding a technology*, and the *documented use cases of the technology*. These elements could all lead to the wrong decisions being made, which would affect the success of a use case. An individual requires a process to follow to combat the effect of biases and measurements and to ensure better decisions are made. To ensure better decision-making, a model is proposed within this study that could help IT practitioners mitigate the effects of biases and measurements. The next chapter will discuss the context of this study.

CHAPTER 2: DECISION-MAKING REGARDING TECHNOLOGY

Chapter 1 argued that decision-making is a difficult process. This dissertation hinges on the problem that IT practitioners do not have a systematic way to select a NoSQL family for a non-arbitrary use case. This research aims to provide such a framework.

The goal of this chapter is to further elaborate on decision-making and the problems that might influence decision-making. This chapter assists in providing context to the rest of the study. This chapter focuses on decision-making and the effects of measurements and biases on decision-making.

2.1 Decision-making

Decision-making is regarded as a problem-solving activity (Kahneman & Tversky, 1979). The activity is completed when a solution which is deemed as satisfactory or optimal is found. A large part of making decisions is to analyse a set of alternative solutions. These alternatives may be seen according to evaluation criteria, where an individual may rank the alternatives according to "attractiveness" (Kahneman & Tversky, 1979). One alternative may be more attractive than another alternative since it meets more criteria. Therefore, the end-goal of the problem-solving activity is to select one of the alternatives to address the problem.

Every individual participates in decision-making on a daily basis, since all tasks require some form of a decision to be made (Hammond, Keeney & Raiffa, 1998; Kahneman & Tversky, 1984). Individuals face decisions from the moment they wake up in the morning until the moment they go to sleep at night. These decisions vary in difficulty and importance. Examples of easy decisions include deciding whether a door should be open or closed or in which direction an individual should walk to reach a certain destination. Most easy decisions do not require a heavy thought process (Hammond et al., 1998; Kahneman & Tversky, 1984).

Many day-to-day activities require an individual to make snap decisions (Hammond et al., 1998). Driving a car requires many snap decisions, as there is little time available to consider and analyse the options in detail. Examples of snap decisions include deciding whether to turn on the lights, switch lanes, change gears, or apply the brakes when stopping is required. Decisions can also be challenging and have a high level of importance connected to their outcomes (Kahneman & Tversky, 1984). Difficult decisions mostly require a cumbersome thought process, since it is important that they lead to the desired outcomes. These types of decisions are made without advanced knowledge of their consequences (Kahneman & Tversky, 1984). An example of a difficult decision is deciding whether an investment is worthwhile or not. There is no certainty about the outcome of such a decision.

Individuals have business goals and objectives that need to be achieved (Clemen & Gregory, 1995). The decisions they make will influence the way in which individuals reach their goals and objectives (Tversky & Kahneman, 1985). When making a decision, each alternative will have different consequences (Kahneman & Tversky, 1979; Kahneman & Tversky, 1984; Tversky & Kahneman, 1985). The consequences of a decision can be advantageous or disadvantageous depending on the situation (Clemen & Gregory, 1995). Therefore, it is important to make the right decisions.

Conflicting business goals can increase the difficulty of making a decision. For example, when deciding whether to implement an expensive technology or an inexpensive technology, a tradeoff is present in both alternatives (Clemen & Gregory, 1995). If the IT practitioner decides to implement the expensive technology, more benefits can be gained from the technology, but there will be fewer finances available for other IT projects. If the IT practitioner implements the inexpensive technology, more finances will be available, but the technology will have fewer benefits.

Different levels of importance are connected to decisions and their outcomes (Kahneman & Tversky, 1979; Kahneman & Tversky, 1984). The more crucial the outcome of a decision, the harder the decision becomes. For example, judging distance requires an individual to make use of *heuristics*, which are small routines based on decisions (Hammond et al., 1998). The heuristic an individual uses for judging distance is clarity times proximity (Hammond et al., 1998). Therefore, the heuristic allow an individual to quickly judge the distance to an object. When an individual is faced with a difficult decision, they can employ heuristics to make a quick decision to solve the problem.

Heuristics apply to technology decisions as well. Along with the importance of the decisions, the amount of information to that needs to be considered also influences IT practioners when making decisions. The more information needs to be considered, the more difficult the decision becomes and the more likely it becomes that using the heuristic will lead to making the wrong decision. This shows that there is a need for a framework that will help IT practitioners make better decisions.

The different levels of risk connected to decisions may also affect the way individuals make decisions (Kahneman & Tversky, 1979; Kahneman & Tversky, 1984). The higher the risks associated with a decision, the more difficult it becomes to make such a decision. If a decision is difficult and requires a heavy thought process, an individual is more likely to experience problems. Two of the problems that individuals can face are the influences of measurement and biases on decision-making.

2.2 Measurement

A *measure* refers to "a standard unit used to express size, amount, or degree" (Oxford English Dictionary, 2017). Therefore, to measure means to "ascertain the size, amount, or degree of (something) by using an instrument or device marked in standard units" (Oxford English Dictionary,

2017). Measuring values can heavily influence the decisions of IT practitioners. If the wrong values are presented through measurement, then bad decisions can be made.

The process of measuring involves "rules for assigning numbers to objects to represent quantities of attributes" (Nunnally, 1967 as cited by Churchill Jr, 1979). The definition of measurement provided by Nunnally (1967) states that measurement involves measuring the attributes of objects, not the objects themselves (Churchill Jr, 1979). However, the measurement definition does not specify the rules through which measurement values are assigned (Churchill Jr, 1979). Measurement involves two rules. The first is that symbols must be assigned to objects. The second is that objects must be classified according to a specific attribute (Nunnally & Bernstein, 1994).

The term *rules* indicate the methods to be used. This needs to be explained in further detail. For a measure to be standardised, the rules need to be clear and practical to apply. The rules should not require great skill from the administrator and the results of the measurement should not depend on the administrator. The use of the term *attributes* within the definition indicates that a measurement focusses on the features of an object and not the object itself (Nunnally & Bernstein, 1994, p. 4). When employing measurement, there are certain concepts that need to be considered.

2.2.1 What to consider when measuring

Making a measurement means determining the value of some quantifiable item (SASO, 2006). Before measurement can occur, enough detail regarding the items to be measured, the method, and the measurement procedure must be provided (SASO, 2006). There are various concepts to consider when measuring values such as how values are aggregated, the precision, and accuracy of values. Other concepts include *uncertainty*, the *mean* (average), the *median*, the *mode*, and *outliers*. These concepts can influence the type of measure to be selected. Just consider mean and mode as an example.

Specific measures are seldom useful. For example, the time it took for a process to complete may be operationally useful, but in the context of making decisions provide much more meaning when the times are aggregated in a specific manner. The average (mean) of the times could be an indication of overall performance (Dean & Dixon, 1951; Gravetter & Wallnau, 2011). However, should there be extreme cases or an uneven distribution, the average may not be a good indication anymore. and other measures such as the mode, which is the middle value of the sorted sample (Gravetter & Wallnau, 2011), may be more appropriate.

Also, to consider is the issue of precision versus accuracy. Accuracy refers to how well the measurement represents the actual value, while precision speaks to the consistency achieved through multiple measurements (Hubbard, 2011, p. 133). Accuracy and precision are not related. A wrongly

calibrated measurement instrument can consistently give the same inaccurate reading, this having high precision and low accuracy. It is also important to consider to which degree accuracy is important. Numbers may have a psychological effect on people. For example, giving a response rate to a questionnaire as 71.62% while mathematical accurate and precise (at least to two decimal places) when you have 53 of the 74 people responded it might provide an inflated sense of security in the measurement. Similarly, highly precise values may be construed as accurate just because they are to the 5th decimal.

The above examples show that there may be more to take note of when dealing with measures than what is immediately apparent. Therefore, special attention must be placed on which type of measure is used and for which purpose it is used. The following section discusses the types of measures.

2.2.2 Types of measures

There are two main categories of measures, namely *nonmetric* and *metric* measures. The nonmetric category is concerned with differences in type or kind that indicate the presence or absence of attributes in subjects (Hair, Black, Babin & Anderson, 2010). The metric category is concerned with differences in degree regarding a specific attribute (Hair et al., 2010). Measures under the nonmetric category are nominal and ordinal measures. Ratio and interval measures fall under the metric category.

Nominal measures use numbers to identify and represent subjects or objects (Gravetter & Wallnau, 2011; Hair et al., 2010). Nominal measures are also known as categorical scales and can be used to decide whether two objects are equivalent or not for categorising purposes (Nunnally & Bernstein, 1994). When there are only two options, for example male or female, one (male) is assigned the number 1 and the other (female) is assigned the number 2. The numbers are used only to keep track of the different categories (Nunnally & Bernstein, 1994) and do not refer to any mathematical calculation to be done (Hair et al., 2010). Therefore, nominal data refers only to the category and not to the quantity of an attribute (Hair et al., 2010; Nunnally & Bernstein, 1994). Categories may not reflect any quantitative relationship, but they can lead to valuable insights concerning correlations within and between the categories (Nunnally & Bernstein, 1994).

Ordinal measures involve a rule where respondents decide whether one subject is greater than or less than the other subjects (Hair et al., 2010; Nunnally & Bernstein, 1994). The subjects can be arranged in order with regards to how much of an attribute the subject possesses (Gravetter & Wallnau, 2011; Hair et al., 2010; Olivier, 2009). Numerical values are assigned to the subjects but have no mathematical meaning. Therefore, they represent the relative position in the order of subjects (Hair et al., 2010). For example, when arranging the names of individuals according to their height from

tallest to shortest (Nunnally & Bernstein, 1994), the order of individuals does not explicitly indicate the differences in height. Therefore, the types of analyses to be performed are limited, as no arithmetic operations can be performed (Hair et al., 2010).

Ratio measures (Olivier, 2009, p. 83) require the respondent to choose from a provided list of ratios. In ratio measures, there exists a true zero (0), which means nothing, rather than an arbitrary zero, which means the middle point (Gravetter & Wallnau, 2011; Nunnally & Bernstein, 1994). Ratio measures can permit all mathematical operations (Hair et al., 2010). An example of a device that uses a ratio measure is a bathroom scale that is used to measure weight. Bathroom scales employ a ratio measure with a true zero to indicate the weight of an individual. The weight values can also be seen in terms of multiples. For example, 50 kilograms equals half the weight of 100 kilograms (Hair et al., 2010).

Interval measures (Gravetter & Wallnau, 2011; Olivier, 2009, p. 83) resemble ratio measures but do not have a true zero. Interval measures provide the user with the ability to perform any mathematical operation using their values (Hair et al., 2010). Interval measures use constant units of measurement to ensure that the difference between any two adjacent points is equal (Hair et al., 2010; Nunnally & Bernstein, 1994). The range of values must have equal intervals between them and the number of values used must have a neutral point. This type of measure is commonly combined with a Likert scale. A Likert scale with a range of 1–5, 1–7, or 1–9 can be used, since these all contain a neutral point. An example of using this type of measure is asking a respondent to indicate what the likelihood of a storm occurring is.

Table 2.1: Measurement scales retrieved from Nunnally and Bernstein (1994).

Measure	Basic operation	Permissible transformations	Permissible statistics	Examples
Nominal	= vs. ≠ (equality vs. inequality)	Any one-to-one	Numbers of cases, mode	Telephone numbers
Ordinal	> vs. < (greater than vs. less than)	Monotonically increasing	Median, percentiles, order statistics	Hardness of minerals, class rank
Ratio	Equality of ratios	Multiplicative (similarity) x = bx	Geometric mean	Temperature (Kelvin)
Interval	Equality of intervals or differences	General linear x = bx + a	Arithmetic mean, variance, Pearson correlation	Temperature (Celsius), conventional test scores

The role of measurement in decision-making can influence the decisions an individual makes. However, it is not the only problem that can lead to bad decisions being made. Decision-making can also be negatively affected by biases.

2.3 Biases

Biases are difficult to define because they are context dependant. However, *The Oxford English Dictionary* (2017) defines bias as "an inclination or prejudice for or against one person or group, especially in a way considered to be unfair" meaning biases refer to prejudiced beliefs or viewpoints. Biases lead to certain subjects being perceived as superior or inferior to other subjects. Therefore, biases in decision-making refer to prejudices that support one decision above other decisions (Hahn & Harris, 2014). A specific outcome is seen as superior to the other possible outcomes of a decision. Thus, if an individual is a victim of a decision-making bias, the alternatives not supported by the bias may not even be considered.

Individuals that are faced with tough decisions tend to employ heuristics (Hahn & Harris, 2014). Heuristics are shortcuts or small routines that individuals can use to make quick judgement calls based on the decision or task at hand (Hahn & Harris, 2014; Hammond et al., 1998). Heuristics help individuals in everyday life. They allow us to make quick judgement calls, be effective, and not waste time (Hahn & Harris, 2014). For example, individuals with heart problems can make use of a heuristic and quickly decide to drink heart medication to prevent having a heart attack.

Heuristics are commonly employed when making complex decisions (Hammond et al., 1998). However, heuristics are not fail-proof and can have a considerable influence on decisions made by individuals. The more complex a decision, the more an individual relies on heuristics to make judgement calls that may occasionally be wrong (Hahn & Harris, 2014). If an individual does not have enough information gathered to effectively employ a heuristic, a biased decision may be made. Therefore, heuristics can introduce biases to individuals making difficult and crucial decisions (Hahn & Harris, 2014; Hammond et al., 1998; Kahneman, 2000).

As mentioned before, making a decision is a planned process that results in a commitment to a proposition (Gold & Shadlen, 2007). When individuals have crucial decisions to make, they typically try to gather as much information as possible. The task of gathering information may be easy or difficult, since individuals may or may not have access to appropriate information. Decisions based on inadequate information cause uncertainty about decisions and their outcomes (Clemen & Gregory, 1995). Thus, uncertainty depends on the state of the knowledge an individual possesses (Clemen & Gregory, 1995). If an individual has sufficient information to know what the outcomes of decisions will be, the individual is certain (Clemen & Gregory, 1995; Hammond et al., 1998). If an individual does not have sufficient information to predict the outcomes of a decision, the individual is uncertain. Uncertainty has a drastic influence on the decisions individuals make. Uncertainty and heuristics introduce biases when complex and difficult decisions must be made (Clemen & Gregory, 1995; Hahn & Harris, 2014).

The example of a judge or jury deciding on a verdict in court illustrates the importance of certainty when making decisions. The judge or jury must gather and investigate all the evidence and identify all possible alternative interpretations thereof before making a decision. They must have certainty about the facts before a verdict can be reached. If there is uncertainty, a wrong and biased decision could be made.

The higher the risks associated with a decision, the more susceptible an individual becomes to biases in decision-making. Multiple biases regarding decision-making have been identified in research (Appendix A). These biases can add to one another and increase the number of flaws in the decision-making process. Biases have been identified in several research areas, such as the financial and behavioural sciences. However, not much research has been done in the area of technology and how biases affect technology decision-making.

2.4 Biases in technology decision-making

There is a large body of literature on biases in decision-making. Appendix A presents some of the research into this topic and lists some biases. Research has been done mainly in areas such as behavioural decision-making, behavioural economics, and managerial decision-making. Behavioural decision-making investigates the choices made by individuals and why these choices were made. Behavioural economics involves psychological insight into the behaviour of humans where economic and financial decision-making are concerned. Managerial decision-making investigates the decisions of top-level management within an organisation. There is sufficient information regarding each bias and how it affects individuals' decision-making. There are several real-life examples of how each bias affects individuals' day-to-day lives, work lives, and financial decision-making. However, research done on how these biases affect IT staff within the technology environment is not common. Thus, for this study, each bias was investigated within a technology context to make an objective decision regarding if and how each bias affects technology decision-making.

Each bias was investigated through literature to achieve an understanding of the bias and what it entails. A definition of each bias was found, and sufficient information was gathered to identify the areas of research in which each bias is generally investigated. Once the research areas were identified, scenarios were formed to illustrate the effects of each bias on decision-making. The researcher then created a technology scenario for each bias to objectively evaluate its effects on technology decision-making. Each bias was assigned a rating out of five stars (*) based on the effect it has on technology decision-making. The rating refers to the applicability of each bias to the technology decision-making context. Figure 2.1 is a graphical representation of the process followed to assign a rating to each bias.

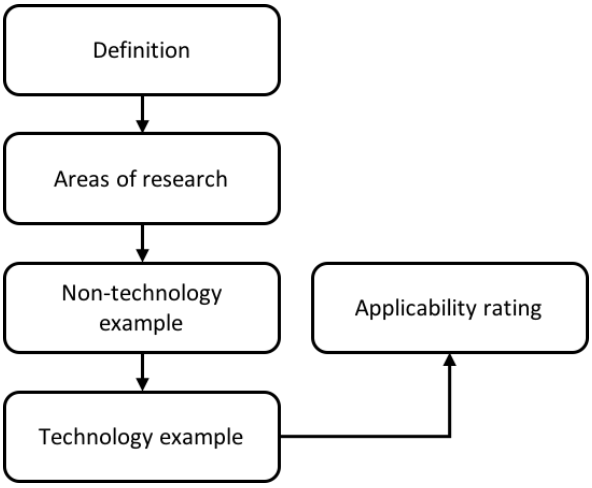

Figure 2.1: Process of rating biases in the area of technology.

After careful investigation and consideration, the researcher identified the biases listed in Table 2.2 as the most relevant to the technology environment. The biases with an applicability rating of five stars (*) are discussed next to illustrate their applicability to the technology environment. Refer to Appendix A for a full list of the ratings applied to the investigated biases.

Table 2.2: Biases affecting technology decision-making.

Bias	Meaning
Status quo	Preferring the current status of a situation and disregarding alternatives
Anchoring	Assuming the first available information is correct and disregarding the alternatives
Sunk cost	Decisions are made to justify past decisions, even when those past decisions are not valid anymore
Confirming evidence	Seeking only information that agrees with established views and disregarding alternatives
Framing	Overemphasising the wrong aspects of a problem
Prudence	Making overcautious decisions based on perceived low risk and disregarding risky alternatives
Recallability	Past experiences heavily influence decisions and current information is disregarded
Shooting from the hip	Making decisions without a systematic decision process
Failure to audit decision process	Not questioning or investigating which decision process to follow
Halo effect	Decisions are based on a single attractive aspect, while the rest of the information is disregarded

2.4.1 Status quo

The status quo bias refers to a bias individuals have regarding the current state of a situation or business decision. The status quo phenomenon has been investigated thoroughly in the areas of economics (Kahneman, Knetsch & Thaler, 1991), managerial business decision-making (Bazerman & Moore, 2008), business decision-making (Dobelli, 2013), and daily decision-making (Hammond et al., 1998). The following paragraph gives an example of status quo phenomenon in daily life.

An individual receives the wine list of a restaurant (Dobelli, 2013) and needs to choose a wine to order. There are a wide variety of wines to choose from. However, the individual is immediately biased and chooses the house wine. Upon consecutive visits to the restaurant, the individual selects the house wine each time. Therefore, the individual is biased towards a single recommended wine. This is an example of the effects of the status quo bias on daily decision-making.

The status quo bias is also relevant to technology decision-making. Individuals in a technology context can also support the current state of affairs and have a bias against the alternatives because they require too much effort. For example, a certain NoSQL database is set up. Switching to an alternative database would involve effort. Therefore, decision-makers affected by the status quo bias will choose to continue using the current NoSQL database. Such decision-makers could potentially reject other NoSQL databases that are better suited for their use cases, because they already have a NoSQL database set up and setting up a new NoSQL database would require too much effort. This shows that individuals in a technology context can also be negatively affected by the status quo bias when making decisions.

2.4.2 Anchoring

The anchoring bias leads to situations where business decisions are based on the first piece of information gathered. The anchoring phenomenon has been investigated in the fields of business decision-making (Dobelli, 2013), daily decision-making (Hammond et al., 1998; LeBoeuf & Shafir, 2006), and heuristics (Tversky & Kahneman, 1974). The research regarding anchoring investigates the effect this specific bias has on the behaviour and decisions of individuals. The following paragraph provides an example of the anchoring bias in daily life.

For example, a marketer attempting to project the number of sales for the coming year (Hammond et al., 1998) may begin by investigating the sales volumes of previous years. The marketer is susceptible to the anchoring bias and the number of sales made in previous years could become an anchor. The marketer will still adjust the previous numbers according to other factors. However, too much weight can be assigned to the past numbers and too little weight to the other factors. Thus, the anchoring bias can influence the decisions of the marketer.

The anchoring bias is also relevant to technology decision-making. Technology decisions can also be made based on the first piece of information gathered, while other factors are ignored. For example, individuals may decide to use the first NoSQL database they hear of, even if it does not fit their use case. In this case, it is the influence of anchoring bias that leads to the decision to implement a specific NoSQL technology without considering the alternatives.

2.4.3 Sunk cost

The sunk cost bias leads to current business decisions being made to justify past business decisions, even when these past decisions are not valid anymore. The sunk cost phenomenon has been explored and described in the areas of economics (Kahneman et al., 1991; Thaler, 1980), business decision-making (Dobelli, 2013), and daily decision-making (Hammond et al., 1998). The sunk cost bias has been thoroughly investigated in the contexts of human behaviour and financial decision-making.

An example of a financial decision that can be affected by the sunk cost bias is deciding to make an investment. Decisions regarding financial investments can be a difficult to make due to the financial risks involved. For example, an individual chooses to invest R100,000 in a specific company by buying shares in that company. The share value drops, and the individual's investment is now worth R60,000. The individual needs to decide whether to sell or keep the shares. In such a situation, the individual can be influenced by the sunk cost bias and decide to keep the shares in the hope that they will increase in value and avert financial loss. Owing to the influence of the sunk cost bias, the individual will not consider the alternative of selling the shares and reinvesting the money in other companies.

The sunk cost bias is also relevant to technology decision-making. The sunk cost bias can lead to additional money being spent on technology that is not relevant anymore. Management may refuse to implement a new technology due to the costs of implementing the current technology. For example, IT staff have spent time, money, and effort implementing a certain NoSQL database. However, this database does not provide the needed levels of support and performance for a specific use case. There are alternative technologies available, but owing to the sunk cost bias, these are not considered for the use case. Thus, the financial burden of maintaining the inappropriate technology is not averted.

2.4.4 Confirming evidence

The confirming evidence bias leads individuals to look for information that endorses or supports their current views and knowledge. The confirming evidence phenomenon has been investigated in the fields of business decision-making (Dobelli, 2013), daily decision-making (Brenner, Koehler & Tversky, 1996; Hammond et al., 1998), important and risky decision-making (Kahneman & Tversky, 1979), and intuitive judgement (Morewedge & Kahneman, 2010).

In day-to-day decision-making, the confirming evidence bias can be a prevalent factor in many decisions. For example, an individual must decide whether to expand a house by adding another room. Currently, building materials are available at a competitive price. However, the individual is experiencing doubts about the cost of the expansion. The neighbours expanded their house six

months ago, when building materials were expensive. The individual decides to get the neighbours' opinion regarding such an expansion, and the neighbours explain that their expansion was expensive. The individual is now influenced by the confirming evidence bias. Since the individual already had doubts about the cost, the confirming evidence leads to a decision not to expand because an expansion would be too expensive.

The confirming evidence bias can also be found in technology decision-making when individuals assume that the information collected is correct and accept that information without considering the alternative. Individuals look for reasons to accept the information they already have without question and do not consider opposing information. They look for confirming information that is in agreement with their pre-existing knowledge. For example, an individual who must choose a NoSQL product to employ asks the opinion of another individual. If the second individual's opinion confirms the knowledge or viewpoint of the first individual, the confirming evidence bias will influence the decision. Alternative opinions will be ignored, which could lead to the incorrect NoSQL database being chosen. Therefore, a bad decision may be made because of the influence of the confirming evidence bias.

2.4.5 Framing

The framing of a question or problem can influence the way a decision is made. The framing phenomenon has been investigated in the areas of business decision-making (Dobelli, 2013), daily decision-making (Hammond et al., 1998; Tversky & Kahneman, 1985), and judgement formation (Strack, Martin & Schwarz, 1988). The framing bias can have an influence on various everyday decisions, including purchasing decisions. For example, two types of meat are found in a grocery store. They are described as being 99% fat-free and containing 1% fat, respectively (Dobelli, 2013). Individuals may perceive the one type of meat (99% fat-free) to be healthier than the other type of meat (1% fat). However, there is no difference in their fat contents, as they both contain the same percentage of fat. The only difference lies in the framing of the information. Thus, individuals may be influenced by the framing bias when choosing between these two types of meat.

The framing bias can heavily influence technology decision-making. If emphasis is placed on the wrong aspect of a problem or question, an incorrect solution could be found. The framing bias can influence the success or failure of a project. For example, overemphasising the relationships between data, even though there is no heavily linked data present, may lead individuals to decide on a technology that can accommodate heavily linked data rather than one that fulfils the other requirements. Thus, individuals can be affected by the framing bias when selecting a technology. If a problem is framed incorrectly with the bulk of the focus placed on the wrong part of the problem, better alternatives can be ignored.

2.4.6 Prudence

The prudence bias causes individuals to be overcautious when high-stakes decisions need to be made. The prudence phenomenon has been investigated in the areas of business decision-making (Dobelli, 2013) and daily decision-making (Hammond et al., 1998). The research investigates the effect of the prudence bias on individuals' behaviour and decision-making. Individuals faced with the task of making difficult decisions want to be certain of what the outcomes of their decisions will be. Individuals perceive certain decisions as low-risk. However, the outcomes of these decisions can actually be more harmful than those of high-risk decisions. An example of the prudence bias in business decision-making follows.

For example, when market-planners for an automotive manufacturer must make a forecast regarding the number of sales for the following year, they slant their forecast numbers in favour of producing additional automobiles to be sure that there will be enough. The market-planners' decision-making is influenced by the prudence bias. They make the safe, low-risk choice. However, this results in the number of cars produced far exceeding the number of sales predicted, which leads to unnecessary financial losses.

The prudence bias can also be found in technology decision-making. Individuals influenced by the prudence bias can cause a project to fail by making the wrong decisions. Individuals want to be safe by making choices that are perceived as low-risk. Therefore, they do not consider high-risk alternatives that may better fit their use case. For example, a business's employees investigate another business's NoSQL technology implementations and choose to implement the same NoSQL database without considering alternatives. However, they do not get the same results as the other business, as their businesses' requirements differ. Their decision was based on the prudence bias and not on what technology would best suit their particular business.

2.4.7 Recallability

The recallability bias causes past events or dramatic occurrences to influence an individual's decision-making. The recallability phenomenon has been investigated in the areas of economics (Kahneman, Wakker & Sarin, 1997), business decision-making (Dobelli, 2013), and daily decision-making (Hammond et al., 1998). Memories and experiences greatly influence the decisions individuals make. For example, when individuals see a train wreck on the news, the memory of the accident will influence their future decisions regarding whether to travel via train or car to reach a destination. Decisions based on past events are often incorrect.

Recallability is also relevant to technology decision-making and can influence the success of project decisions. An IT practitioner could recall a specific experience with a certain NoSQL database

whenever faced with NoSQL technologies. This could cause the individual to make biased decisions where NoSQL technologies are concerned. If the recalled experience was negative, the individual may not consider implementing that NoSQL technology, even if it best fits the problem. If the recalled experience was positive, the individual may choose to implement that NoSQL technology, even if it does not solve the problem. Thus, biased decisions can lead to technologies that are not appropriate for the specific use case being implemented.

2.4.8 Shooting from the hip

Shooting from the hip is a decision-making bias that prevents individuals from following a systematic decision-making process to make decisions and solve problems. The shooting from the hip phenomenon has been investigated in business decision-making (Russo, Schoemaker & Russo, 1989). This bias can negatively influence decision-making. For example, a mechanic disassembling an engine needs to keep track of where each screw was removed from. An experienced mechanic may feel that they have enough experience disassembling engines to not have to take note of the exact position of each screw. Such a mechanic is influenced by the shooting from the hip bias. The bias results in the decision not to follow the process of noting all facts and relevant information. The mechanic may attempt to rebuild the engine and fail, since some screws do not fit or are placed in the wrong holes. Thus, decisions influenced by the shooting from the hip bias can have negative outcomes.

Shooting from the hip is a bias that is relevant to technology decision-making. Individuals perceive themselves as capable enough not to have to follow a decision-making process. Decision-making processes assist individuals by gathering enough information to base decisions on. If individuals do not follow a process, they can make mistakes or miss key facts, which can lead to bad decisions being made. Individuals confident in their knowledge about certain technologies may be faced with a NoSQL problem. If the individuals believe that they can decide on a solution without investigating all alternatives, they can easily make incorrect decisions.

2.4.9 Failure to audit decision process

Failing to audit a decision process means not questioning or investigating the decision process. If individuals do not follow a decision process, they become vulnerable to many decision biases. If they do follow a decision process, they are still vulnerable to the failure to audit decision process bias. The phenomenon of failing to audit decision processes has been investigated in the field of business decision-making (Russo et al., 1989). It is important to follow a specific process when making decisions.

For example, to build a car from the ground up requires an individual to follow a decision process with specific steps. If the steps are mixed up, the car parts will not fit together correctly. An individual that does not question their decision process is susceptible to the influence of the failure to audit decision process bias. This means that even though an individual is following a decision process, it may not be the right decision process to solve the problem.

The goal of a decision process is to assist an individual in understanding the problem, the various decisions that can be made, and how to implement those decisions. It is crucial to consider the failure to audit decision process bias in technology decision-making. If an individual follows the incorrect decision process, the individual may make bad decisions. When following a decision process to choose a NoSQL technology, the use case problem needs to be understood first. Thereafter, the choices of NoSQL storage technologies should be identified. Finally, the implementation process of the chosen NoSQL technology must be understood. If an individual does not follow the correct process, the problem can be misunderstood, and the wrong technology chosen.

2.4.10 Halo effect

The halo effect refers to the way one specific aspect of an item can affect how individuals perceive the item as a whole. The halo effect phenomenon has been researched in the field of business decision-making (Dobelli, 2013). An example of the halo effect can be found in marketing campaigns that advertise products using celebrities. For example, Roger Federer, a professional tennis player, appears in an advertisement for a coffee machine (Dobelli, 2013). Some individual viewing this advertisement may feel the need to purchase that specific coffee machine because their favourite tennis player is endorsing it. Their decision to purchase the coffee machine is based on a single element – the famous tennis player who is endorsing it – and other factors, such as the reliability and usability of the coffee machine, are ignored. Thus, they are experiencing the halo effect.

The halo effect bias is also relevant to technology decision-making. If a NoSQL technology has a certain property that is appealing or unappealing to individuals, they may make biased decisions based on a personal feeling regarding the technology. If the feeling is positive, they may choose this technology over others, even if it is not the best choice. If the feeling is negative, they may not consider choosing the technology, even if it is the best solution to the problem. Thus, the decision is based on a single element and the alternatives are not investigated.

2.5 Four categories of biases

The previous sections showed that there are a wide variety of biases that can affect decision-making. Benson (2016) grouped these biases into four main categories according to the overarching problem faced by each group. The four categories are *too much information*, *not enough meaning*, a*ct fast*, and *memory effects* (Benson, 2016). Each kind of problem occurs for a reason. Too much information

problems occur when an individual is overwhelmed by the amount of information that needs to be considered. Not enough meaning problems occur when the information that has to be considered has no context or meaning. Act fast problems occur when individuals react too fast and ignore crucial pieces of information. Memory effects problems occur when the human memory cannot keep up with the volume of information.

Table 2.3: Bias categories adapted from Benson (2016).

	Categories	Biases
1	Too much information	Anchoring
		Confirming evidence
		Framing
2	Not enough meaning	Failure to audit decision process
		Halo effect
3	Act fast	Status quo
		Sunk cost
		Prudence
		Shooting from the hip
4	Memory effects	Recallability

To overcome these categories of biases, certain methods are required. To counter the first category of biases, a method is required to provide only presently relevant information. This will ensure that less information needs to be considered. To counter the second category of biases, a method is required to provide only contextual information. This will ensure that the information has meaning. To overcome the third category of biases, a method is required to force individuals to follow steps to reach a conclusion. This will ensure that the individual does not react too fast. To overcome the last category of biases, a method is required to encourage individuals to investigate specific information before reaching a conclusion. This will ensure that the human memory is not overloaded with information. Mitigating the effects these four categories of biases have on decision-making can lead to better decisions and outcomes.

2.6 Decision-making techniques

As stated above, there are numerous decisions with varying degrees of difficulty. The more difficult a decision is, the more thought is required to make the decision. To assist with making difficult decisions, one can use decision-making frameworks (Rokach & Maimon, 2005). One example of a often used decision framework is that of a decision tree. The goal of a decision framework is to manipulate the alternatives to assist in making the optimal decision. A framework provides a structured format to think about a decision and all its alternatives.

Decision-making frameworks are able to assist individuals by bringing structure to the decision-making process (Newton, 2016). There exists a wide variety of frameworks which an individual can choose from to assist in making decisions. Example frameworks are the Kepner-Tregoe Matrix,

Decision Matrix Analysis, The Analytic Hierarchy Process, Pareto Analysis, The Futures Wheel, and Force Field Analysis (Newton, 2016).

To illustrate the point of decision frameworks, consider two examples. Fisrt, the Kepner-Tregoe Matrix as the framework of choice. This framework employs four steps which an individual can follow from the start to the end of the decision process (Lumsdaine & Lumsdaine, 1994). The first step is 'situational analysis' where the top-level view of the decision to be made is gained (Newton, 2016; Scheubrein & Zionts, 2006). The second step is 'problem analysis' where the problem is investigated to identify the root cause of the problem (Newton, 2016; Scheubrein & Zionts, 2006). The third step is 'decision analysis' where the alternatives are investigated and evaluated to find the solution to the problem (Newton, 2016; Scheubrein & Zionts, 2006). The last step is 'potential problem analysis' where the selected solution is analysed to identify additional problems that could occur from making such a decision (Newton, 2016; Scheubrein & Zionts, 2006). If an individual follows the steps of the framework, then he/she are enabled to make a good decision.

Second, consider decision trees. A decision tree represents a decision and its consequences, that is the subsequent decsions that must be made. The tree consists of a node, also known as the root, which has no incoming edges (Magee, 1964; Rokach & Maimon, 2005). The root can be viewed as the main issue or problem. All subsequent nodes, known as 'leaves' or decision nodes, have outgoing edges which are known as test nodes. Each of the decision nodes can be seen as subsequent decisions to be made as a result of the root problem. Each decision node splits the instance into two or more sub-instances according to the input and test of a function (Rokach & Maimon, 2005). As a result, a decision tree can be employed to map out a decision and its alternatives to assist with making decisions (Magee, 1964).

As a result of the above mentioned, the need for frameworks in decision-making is clear as they can improve the quality of decisions individuals make.

2.7 Conclusion

In this chapter, it was established that decision-making is a daily task that all individuals partake in and that decision-making can be easy or difficult. It was also established that *measurements* and *biases* can negatively affect decision-making and cause individuals to make bad decisions. There are a wide variety of measurements and biases. The effects of these measurements and biases need to be mitigated to ensure better decision-making.

This chapter elaborated on the context of this study to indicate the need for a model to assist IT practitioners with technology decisions. Since IT practitioners do not have a systematic way to decide between NoSQL families, a need for a way to make such decisions without being affected by the

problems of measurement and biases exists. However, before such a model can be created, NoSQL must first be understood. Therefore, the next chapter will investigate NoSQL technologies.

CHAPTER 3: NOSQL

The second chapter focused on explaining how difficult decision-making is and that it is a daily task. It also established that measurements and biases can negatively influence decisions and increase the difficulty of decision-making. Therefore, there is a need to mitigate the effects of biases and measurement on technology decisions.

The technology context for this study is found in NoSQL (Not only SQL). Therefore, the chapter starts by investigating data storage technologies in general. Thereafter, a comparison is made between relational and non-relational (NoSQL) storage technologies. The data model behind NoSQL storage technology is an important aspect that needs to be considered. There are four types of data models used by NoSQL technologies that need to be investigated. Decision-making regarding NoSQL is further complicated by this.

3.1 Storage technologies

A database is a technology used for storing data. Data can originate from various sources and have different formats. Numerous types of data storage technologies, such as hierarchical databases, network databases, relational databases, and database management systems (DBMS), have been developed (Naheman & Wei, 2013). Currently, the two terms that are generally used to differentiate between types of database technologies are *relational* and *non-relational* databases.

The creation of relational databases has been key to the development of storage technologies (Naheman & Wei, 2013). However, the development of technology and the internet contributed to the creation of large volumes of semi- and unstructured data. Relational databases struggle to store such data. Storage technologies should not struggle to accommodate semi- and unstructured data (Naheman & Wei, 2013). Continuous growth in technology means that data storage technologies also need to evolve to keep up with the ever-growing requirements to accommodate different types of data (Leavitt, 2010). These new requirements are based on the data model, the data storage, and the distributed architecture (Naheman & Wei, 2013). Firstly, the *database* refers to the physical storage of data. Secondly, the *database systems* refer to the management software to manage the database, and thirdly, the *database model* refers to the representation of data and how it is stored.

3.1.1 Relational databases

A relational database is a set of tables containing data that is fitted into predefined categories (Leavitt, 2010). The data have relations to one another, hence the name relational database (Han, Cai & Cercone, 1993). Each table in the relational database contains one or more categories of structured and organised data that is stored in rows and columns (Han, Cai & Cercone, 1993). The relational data structure allows information from different tables to be linked (Padhy et al., 2011). The table

should also have a key, which is used to uniquely identify the data and to create links between the tables (Padhy et al., 2011).

The relational model also employs a procedure called *normalisation*. Normalisation is a set of procedures used to remove redundant values and improve data integrity (Chapple, 2018; Coronel & Morris, 2016). In doing so, normalisation prevents data manipulation irregularities and loss of data integrity. A large advantage to database normalisation is that data is logically stored, and less space is consumed (Chapple, 2018). The way to conduct normalisation is through the processes called 'normal forms' which has five forms namely, 1NF to 5NF (Coronel & Morris, 2016). Each of these forms can be seen as increased levels of normalisation. After 1NF, each subsequent form must meet the requirements of the previous form with additional requirements (Chapple, 2018).

Some benefits of normalisation include better overall database organisation, reducing the volume of redundant data, higher consistency, and possible better database security (Stephens, Plew & Jones, 2009). However, there is one major drawback to normalisation, namely reduced database performance. The reason is because more resources are required to react to a query since the data must be located, joined from multiple tables, and then processed to provide an answer to the query (Coronel & Morris, 2016; Stephens, Plew & Jones, 2009). Therefore, database normalisation has an impact on the transactions within the relational database.

Most relational database systems are based on transactions to ensure the integrity of the data. Transactions ensure the *atomicity*, *consistency*, *isolation*, and *durability* (ACID) of data management (Moniruzzaman & Hossain, 2013). Atomicity refers to the ability of the database management system to follow an 'all or nothing' approach to transactions (Yu, 2009). Therefore, if a part of the transaction fails, then the whole transaction fails. Consistency refers to the degree of consistency of the database after a transaction is completed or failed (Medjahed, Ouzzani & Elmagarmid, 2009; Yu, 2009). Therefore, the focus would be on the state of the data and whether it is in a consistent state before and after the transaction occurs. Isolation means that the data used in the transaction cannot be accessed by other processes during the transaction (Medjahed et al., 2009; Yu, 2009). Durability refers to the guarantee that transactions will persist even if system failures occur (Medjahed et al., 2009; Yu, 2009). Therefore, the transaction will not be affected by system failure.

As a result of the above, ACID aims to provide assurance and guarantee the reliability of database transactions (Moniruzzaman & Hossain, 2013). ACID works well with structured data such as banking transactions. However, when faced with unstructured data, it may struggle to provide guarantees on the reliability of the data.

Therefore, relational databases operate best when handling structured data and struggle to accommodate semi- or unstructured data (Leavitt, 2010; Zhang, 2011). Another solution was found to accommodate the drawbacks of the relational data model which is a non-relational technology known as NoSQL (Naheman & Wei, 2013). There are certain limitations and drawbacks to the relational data model when it is faced with storing semi- or unstructured data. The limitations need to be highlighted and understood to indicate why non-relational (NoSQL) databases are a better solution when storing or working with semi- or unstructured data.

3.1.2 Limitations of relational databases

Scaling, *complexity*, *large feature set*, and *slow reading and writing performance* are some of the limitations of relational databases when working with semi- and unstructured data (Han, Haihong, Le & Du, 2011; Jatana et al., 2012; Leavitt, 2010). These include scaling, complexity, large feature set, and slow reading and writing performance.

S*caling* refers to increasing the database size. There are two methods of scaling a database can employ, namely vertical and horizontal scaling. Vertical scaling involves increasing the storage and processing capacity, which can become a costly venture. Horizontal scaling means storing the same table across multiple servers (Han et al., 2011; Jatana et al., 2012; Leavitt, 2010). Therefore, when storing large volumes of semi- and unstructured data, it would be better to use *horizontal* scaling, as this could be a cheap method (Jatana et al., 2012; Leavitt, 2010). However, horizontal scaling can be expensive also.

Complexity refers to the ease of working with the data structure. Relational databases work with a fixed structure (Leavitt, 2010). Therefore, if the data does not fit into the structure, conversion and change of the data must occur.

Relational databases can also offer a *large set of features*, which can complicate the work to be done (Leavitt, 2010). These additional features also add to the total cost of the relational database (Jatana et al., 2012). Thus, the database system could turn out to be very costly due to having additional features that may not even be needed.

Slow reading and writing performance may be found when relational databases are used to work with semi- and unstructured data (Han et al., 2011; Leavitt, 2010) which indicates that relational databases cannot handle the semi- or unstructured data.

Non-relational databases were created to overcome the limitations of relational databases. Therefore, non-relational databases are focussed on working with or storing semi- and unstructured data. The development of non-relational data storage technologies has led to the creation of NoSQL.

3.1.3 Non-relational databases

Non-relational databases refer to a data model that differs from relational systems in that data is stored without an explicit structure (Padhy et al., 2011). Non-relational databases do not use tables as a data structure to store data in. Other aspects that differentiate non-relational databases from relational systems are, firstly, that non-relational systems do not just use SQL as their query language. Secondly, non-relational databases do not guarantee ACID properties. Thirdly, they do not employ join operations, and lastly, they can scale horizontally (Jatana, Puri, Ahuja, Kathuria & Gosain, 2012). Non-relational databases are commonly linked with NoSQL because NoSQL is non-relational in nature (Naheman & Wei, 2013).

3.1.4 Overcoming limitations of relational databases

As stated before, relational databases struggle to accommodate semi- or unstructured data. Therefore, NoSQL databases were created to overcome such limitations. The first identified limitation was scaling where relational databases use vertical scaling to scale the data. However, with the increase of datasets and the increasing use of semi- or unstructured data, makes it difficult and expensive to scale with relational databases (Kuhlenkamp, Klems & Röss, 2014). As a result, NoSQL employs horizontal scaling to counter the limitations of vertical scaling by exponentially increasing the capacity and performance to accommodate the size and type of data (Moniruzzaman & Hossain, 2013). However, horizontal scaling can be more expensive to implement than vertical scaling.

The second limitation identified is complexity. Relational databases have a focus on working with structured data which means the complexity is low since the data fits into the structure with ease (Moniruzzaman & Hossain, 2013). However, if semi- or unstructured data are stored within a relational database, then the complexity is high since the data must be manipulated and changed to fit the fixed data structured (Abadi, 2009). NoSQL places fewer constraints on the structure of the data to be stored (Győrödi, Győrödi & Sotoc, 2015a). Therefore, NoSQL allows the storage of several data structures without the need to manipulate the data structures to meet certain requirements.

The third limitation identified is slow reading and writing performance (Naheman & Wei, 2013). Relational databases provide excellent performance when dealing with structured data. However, when semi- or unstructured data is stored, then the performance of the relational database will slow considerably down (Han et al., 2011; Naheman & Wei, 2013). Therefore, NoSQL overcomes this limitation since it can provide high reading and writing performance when working with semi- or unstructured data (Hecht & Jablonski, 2011).

3.2 Classification of NoSQL databases

There are four families of NoSQL databases, namely *key-value stores*, *column-family stores*, *graph stores*, and *document-based stores* (Aniceto et al., 2015). The following section describes and provides a general use case for each family of NoSQL.

3.2.1 Key-value stores

Key-value stores are database management systems that store keys (identifiers) and values associated with the keys inside a hash table (Moniruzzaman & Hossain, 2013). The values can vary from simple text to complex lists. Data searching is done against the keys and looks for exact matches. Figure 3.1 is a graphical representation of a key-value store.

Key-value stores contain data that is being stored in a key to value pair (Aniceto et al., 2015; Padhy et al., 2011). The values stored are indexed for retrieval by unique keys (Aniceto et al., 2015; Padhy et al., 2011). Values are stored independently from each other and the application logic handles the relationships between the data (Aniceto et al., 2015). Key-value stores can handle structured and unstructured data (Padhy et al., 2011). They are simplistic and ideally used when highly-scalable databases that can retrieve and store large volumes of records quickly are required (Moniruzzaman & Hossain, 2013).

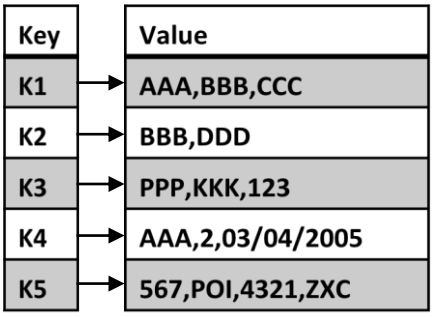

Figure 3.1: Key-value store contents, adapted from Wellhausen (2012).

3.2.2 Use case for Key-Value stores

Key-value databases may be an appropriate solution for applications with one kind of object where queries are based on one attribute (Cattell, 2011). Key-value databases are best suited for use cases such as managing user profiles within a large financial business which requires a scalable and high-performance database (Moniruzzaman & Hossain, 2013). Amazon is making use of a NoSQL database called *Dynamo*. It is a key-value store that is readily available and is used in their shopping cart feature (DeCandia et al., 2007). Implementing a key-value store provides Amazon with a scalable and available distributed data store for their online business.

Another example of a key-value product is *Riak*. A couple of uses for Riak (Nayak, Poriya & Poojary, 2013) include firstly, managing personal information on social media websites. Secondly, to manage profiles for Massively Multiplayer Online Role-Playing Games (MMORPGs). Thirdly, manage factory and information control systems and lastly, to collect Point of Sales (POS) or checkout data. Other products belonging to the key-value family are *Voldemort* (used by LinkedIn), *Redis*, and *Berkeley DB*.

3.2.3 Document-based stores

Document-based stores utilise the same concept as key-value stores (Aniceto et al., 2015). Documents refer to collections of attributes where each attribute can have multiple values assigned to it (Aniceto et al., 2015; Padhy et al., 2011). One can add any number of records of any length to any document in the store (Padhy et al., 2011). Each document in the store contains an ID key that uniquely identifies that specific document (Aniceto et al., 2015). The documents are encoded in a data exchange format such as JSON (JavaScript Option Notation) which are commonly stored in document-based stores. Examples of document-based stores include *MongoDB* and *CouchDB* (Padhy et al., 2011).

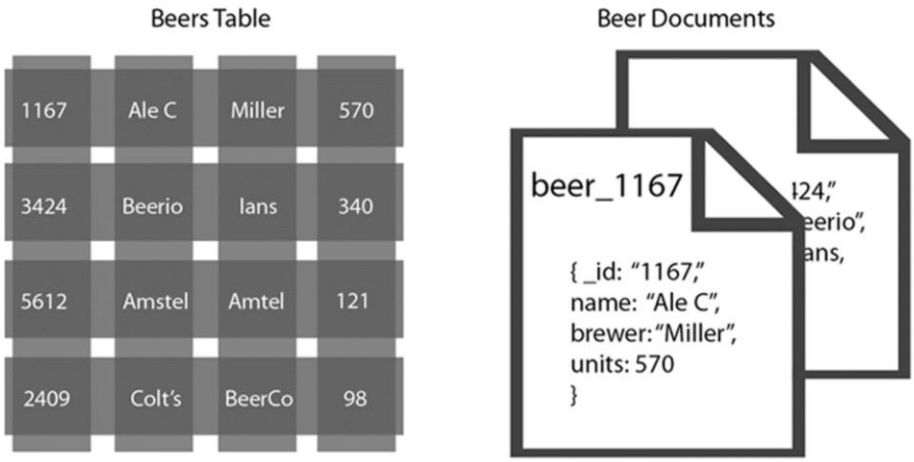

Figure 3.2: Relational data model versus the document-based data model (Couchbase, n.d.).

The ID keys that explicitly identify each specific document must be unique (Hecht & Jablonski, 2011). The values inside a document store can be queried, which means that complex data structures can be handled with ease. Document stores do not have any schema restrictions and adding new attributes to a store is easy (Hecht & Jablonski, 2011). A user can also do multi-attribute searches from a variety of key-value pairs. Document stores are convenient for data integration and schema migration tasks.

3.2.4 Use case for document-based stores

Document-based stores are good for managing and storing enormous size collections of documents, such as collections of text documents, emails, and XML documents, as well as semi- and unstructured data (Moniruzzaman & Hossain, 2013). Document-based stores can be used by content management systems, blogging platforms, and applications for electronic commerce as well as for web analysis and real-time analysis. However, document-based stores may not be an appropriate solution for websites with complex transactions or queries that dynamically change the calculation structure (Hwang, Lee, Lee & Park, 2015). The document data model can facilitate website creation, since the data model supports unstructured data by default while not requiring costly and time-consuming migrations between systems (Hwang et al., 2015). An in-practice example of a document store in use is traffic department records that contain two categories of objects (vehicles and drivers) and can do a lookup of objects on multiple fields (driver's name, owned vehicle, birth date, license number).

An important element to consider is the level of concurrency a task requires when employing document-based databases. If "eventually consistent" can work with the use case, then document-based stores may work well (Cattell, 2011). An example of this idea being applied to daily life can also be found in traffic department records. The traffic department may not need to know if a driver of a specific license number received a traffic violation within the last minute. However, the records will be updated eventually

3.2.5 Graph stores

Graph stores are excellent at handling and managing heavily linked data. Graphs represent the data schema for this database type. In cases where relationship-heavy data is stored, graph data models are better suited to handle the data than other kind of data model (Hecht & Jablonski, 2011). Graph stores consist of three elements, namely nodes, the relationships between nodes, and the values attached to the relationships and the nodes (Aniceto et al., 2015). Graph stores are the only NoSQL database type that focuses on the relationships between the data (Moniruzzaman & Hossain, 2013). Graph stores also visually represent the data, which is more human-friendly.

Graph stores can be used by location-based services and recommendation systems. They can also be used for knowledge representation and to solve pathfinding problems in navigation systems (Hecht & Jablonski, 2011). The use cases, for example pathfinding problems, employ complex relationships.

3.2.6 Use case for graph stores

Graph databases are more useful when the relationships between the data are more important than the data itself (Moniruzzaman & Hossain, 2013). Graph stores are optimised for traversing relationships. The best use cases for graph stores are when dealing with heavily linked data, location-

based services, and recommendation services (Hwang et al., 2015). Graph stores do not provide the best solution for modifying entity updates (Hwang et al., 2015). However, graph stores can be used as a mechanism for graph-based queries, such as computing the shortest path between two nodes inside a cluster. Another possible use is pattern detection through forensic investigation (Moniruzzaman & Hossain, 2013). Examples of graph databases include *Neo4j*, *InfoGrid*, *GraphDB*, and *InfiniteGraph*.

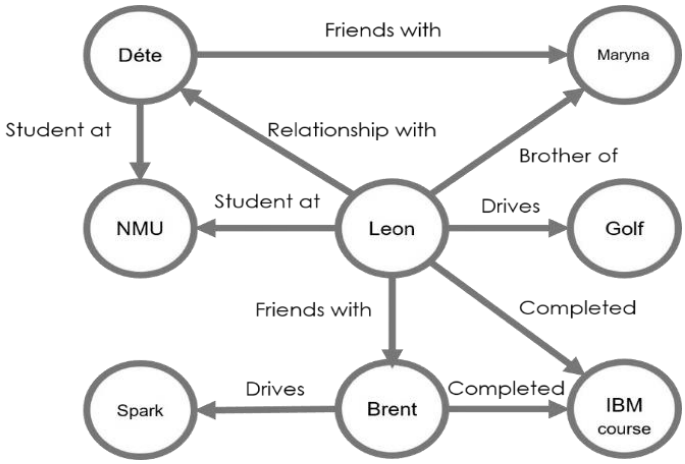

Figure 3.3: Graph NoSQL Database

3.2.7 Column-family stores

Column-family stores define the structure in which the data is stored as a set of columns (Aniceto et al., 2015). Column-family stores contain an extendable column of related data. They can also be referred to as *super columns* or *column-family structures* (Padhy et al., 2011; Aniceto et al., 2015). Columns refer to the data schema of the column-family databases. Examples of products that fall under the column-family category are *Cassandra*, *HBase*, and Google's *BigTable* (Padhy et al., 2011).

Column-family databases can be used for distributed data storage. They can also be used for large-scale, batch-oriented data processing, such as converting, sorting, and parsing data (Moniruzzaman & Hossain, 2013). An example of this is the conversion of numbers between binary and hexadecimal values. Statisticians or programmers can do predictive and exploratory analytics on the data stored in column-family databases (Moniruzzaman & Hossain, 2013).

3.2.8 Use case for Column-family stores

Column-family store use cases focus on multiple kinds of objects and do lookups based on any field (Cattell, 2011). Column-family stores aim to provide higher throughput and concurrency (Cattell, 2011). However, their complexity of use is higher than that of other NoSQL families (Cattell, 2011). Column-family stores are suitable for distributed data storage, large-scale data processing, and exploratory and predictive analytics (Moniruzzaman & Hossain, 2013). Column-family stores can

be used firstly, by content management systems to store all data regarding the contents. Secondly, blogging-platform services to store and enable blogging of events. Thirdly, visitor countering services to keep track of visitors and lastly, for event logging to capture and store the occurrence of specific events (Hwang et al., 2015).

Using this type of store for blogging would allow blog entries to be stored alongside tags, categories, links, and trackbacks in different columns (Hwang et al., 2015). Furthermore, when storing customer information for an online transaction web application, the data must be partitioned vertically and horizontally. This type of database allows customers to be clustered by country and can store data that rarely changes in a different place than data that is changes regularly. This could also be achieved by using document-stores. However, it is more easily achieved by using column-family databases (Cattell, 2011). Examples of products belonging to the column-family of NoSQL technology are *HBase* and *Hypertable.*

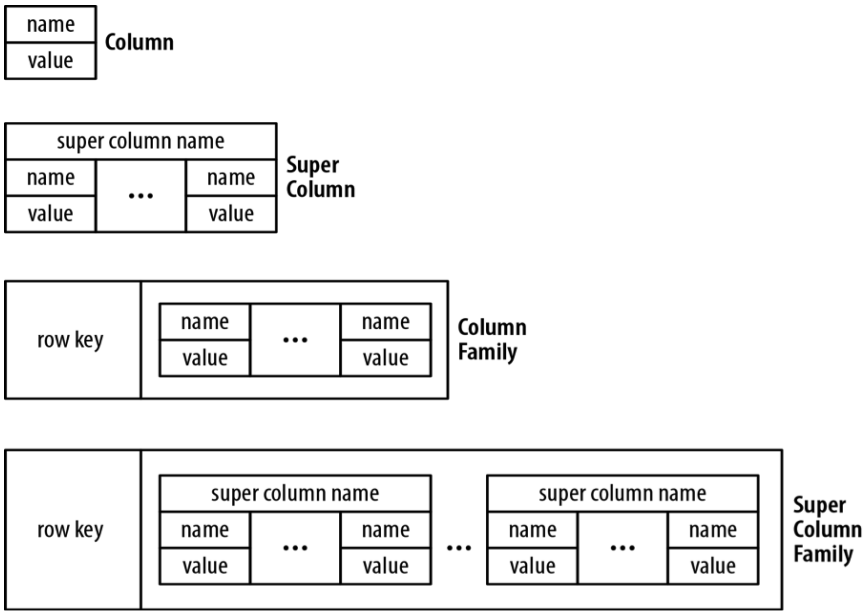

Figure 3.4: Wide-Column Store NoSQL Database (Sasaki, 2015).

3.3 Conclusion

Non-relational storage technologies, which can also be referred to as NoSQL technologies, overcome the limitations of relational storage technologies. This chapter provided background information relating to NoSQL technologies. There are four families within NoSQL and there are advantages and disadvantages to employing each of them. Within each family, there are a variety of products that can be selected and used. When faced with a use case that relational technology cannot accommodate, decisions must be made regarding which NoSQL family to use. The data storage model is a crucial aspect to consider when deciding between NoSQL families and products. A decision must be made regarding the best data model and technology option for the use case. Therefore, it is important to

have a decision model that removes as much ambiguity and provides as much structure as possible to combat the influence of measurements and biases.

Individuals need to do their best to be aware of biases and to prevent these biases from influencing the decisions made. Therefore, the following chapter will investigate and discuss the decision framework that can be used to aid in the decision process.

PART B

FRAMEWORK

CHAPTER 4: CONCEPTUAL FRAMEWORK

The previous chapters have established that decisions regarding NoSQL families are difficult and can be affected by biases and measurements. IT practitioners lack a systematic process to follow when making these types of decisions. Therefore, a need for a process that will enable better decision-making by reducing the effects of biases and measurements exists.

Chapter 4 provides a conceptual framework that aims to minimise the effect of decision biases. The conceptual framework comprises three parts: constructs, a weighted decision model that is formed by combining the constructs, and a process model that explains how to tailor the decision model to specific scenarios.

The chapter starts by motivating the need for the proposed framework and explaining how it would help. Thereafter the chapter develops the three parts of the framework. Firstly, the constructs are identified. Secondly, the weighted decision model is specified. Lastly, the process model is defined.

4.1 Why is the framework necessary?

During the day, many decisions with varying levels of complexity and importance need to be made. The level of complexity a decision can influence individuals' decision-making (Kahneman & Tversky, 1979; Kahneman & Tversky, 1984). When faced with difficult and complex situations, individuals may employ heuristics to make quick judgements (section 2.1) (Hammond et al., 1998). However, heuristics are not always fail-proof and can lead to wrong decisions being made.

IT practitioners are constantly faced with technology-based decisions with varying levels of complexity. An important decision they need to make is which NoSQL family to use when doing a project where NoSQL is used as the storage technology. This decision is not made in an impromptu manner and can be influenced by several biases. Therefore, it is important for IT practitioners to be aware of these biases when making decisions regarding NoSQL. An IT practitioner can counter several decision-making biases by understanding the unique strengths and weaknesses of each NoSQL family.

There are four NoSQL families and numerous products within each of these families (Edlich, 2011). Each family has a data model that is used to stored data. Several products employ the data models of specific NoSQL families. The data models handle stored data in different ways and each data model has its own unique strengths and weaknesses. The unique strengths and weaknesses of each NoSQL family need to be understood before an informed decision can be made regarding which of them to use.

To make an informed decision regarding which family and product to use, IT practitioners need to consider large volumes of information. A framework that aims to mitigate the effects of biases and measurements on decision-making needs to be created. There are two situations that require the assistance of such a framework.

The first situation presents itself when a decision must be made regarding which of the four NoSQL families is best suited for a specific use case. If uncertainty regarding this choice exists, the IT practitioner can employ the framework to counter decision-making biases and make an informed decision.

The second situation presents itself when a decision must be made between NoSQL products. Once a NoSQL family has been chosen, the IT practitioner must decide on a specific product to use for the same use case. The same decision-making biases apply to decisions regarding NoSQL products as to those regarding NoSQL families. Therefore, the IT practitioner can employ the same framework to better decide between NoSQL products.

These are the two most prominent situations the framework will assist IT practitioners with. However, the framework can assist them in other ways too. Current documented use cases cater for the different data types that may be stored within NoSQL technologies. They do not however cater for uncommon types of data stored in NoSQL technologies. Therefore, the proposed framework caters for more than just the type of data to be stored. In addition, the framework also makes the IT practitioner more aware of biases. The framework will result in a better decision-making regarding NoSQL technologies. Therefore, there is a need for the proposed framework.

4.2 How will the framework help?

The proposed framework can assist IT practitioners in making technology-based decisions. It contributes in multiple ways:

- A set of fixed criteria allows uniform comparison.
- A list of options ensures that all relevant options are considered.
- Weights ensure that criteria are considered to the degree that they matter in the use case.
- Grading ensures that all options are carefully considered.
- A decision model suggests a choice based on the calculation of a final score.
- A process model caters for adaptation to specific technologies.

Through the ways mentioned above, the framework aims to assist IT practitioners in making better decisions regarding technology. The proposed framework encompasses several artefacts, which are discussed in the following section.

4.3 What does the framework encompass?

The framework comprises multiple artefacts that aid in the decision process. These artefacts map well to the four levels of artefacts proposed by March and Smith (1995). Figure 4.1 graphically depicts the four levels of artefacts. The first level refers to constructs and the second level refers to a model, which is defined as the relationships between constructs. In the proposed framework, the decision model links various constructs (criteria, weights, grades, options) through the calculation of a final score. The third level of March and Smith's (1995) artefacts refers to a method, which is a set of steps followed to perform a task. In the proposed framework, a 6-step process to adapt the decision model to specific technologies is presented.

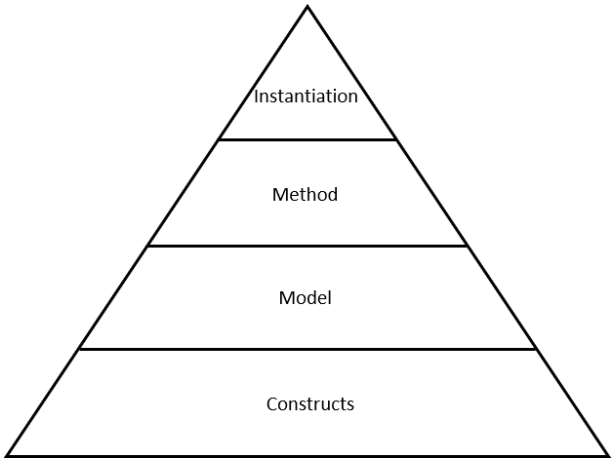

Figure 4.1: Artefacts of a design science study (March & Smith, 1995).

The proposed framework can be used by the IT practitioner to assist with decisions in specific use cases. This use of the model represents the fourth level of March and Smith's (1995) artefacts, namely instantiation.

The following sections discuss the artefacts the proposed framework is comprised of in more detail.

4.3.1 Constructs

There are six constructs found within this framework: the list of choices, the fixed set of criteria, the weights of criteria, the grades, the score calculation, and the sequence of steps.

The first construct in the decision model is the list of choices to decide between. To ensure a uniform comparison, it is important to create a fixed set of criteria, which is the second construct within the decision model. The different criteria are not of equal importance and therefore must be weighted. The weights of criteria is the third construct within the model. Each choice will receive a grade for each of the criteria, which is the fourth construct in the model. A weighted average can be used to do a score calculation, which is the fifth construct in the model. The last construct in the model is the sequence of steps followed to apply the framework.

4.3.2 Model

Table 4.1 represents the weighted decision model with all the constructs inside. The list of choices (families F_1 to F_m) will be graded (R_{11} to R_{nm}) according to the fixed set of criteria (C_1 to C_n). The fixed set of criteria will be assigned weight values (W_i to W_n) that reflect the use case requirements. The decision model will calculate a weighted score ($Score(F_k) = \sum_{i=1}^{n} W_i . R_{ik}$) for each of the families, which can assist in decision-making. To implement this model, a specific method in the form of a process model is required.

Table 4.1: The weighted decision model.

Criteria	Weight	F_1	F_2	…	F_k	…	F_m
C_1	W_1	R_{11}	R_{12}	…	R_{1k}	…	R_{1m}
…							
C_i	W_i	R_{i1}	R_{i2}	…	R_{ik}	…	R_{im}
…							
C_n	W_n	R_{n1}	R_{n2}	…	R_{nk}	…	R_{nm}

$$Score(F_k) = \sum_{i=1}^{n} W_i . R_{ik}$$

4.3.3 Method

Figure 4.2 is a visual representation of the process model found within the framework. The process model provides a 6-step process to implement the framework. Following these steps can also assist with NoSQL product recommendations.

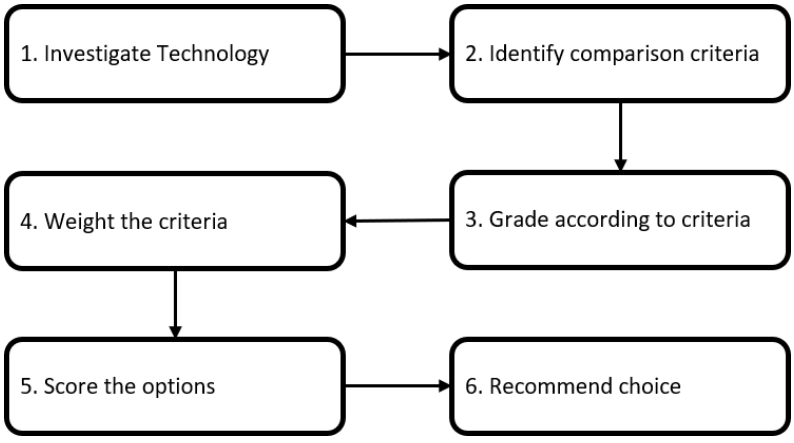

Figure 4.2: General steps of the weighted decision model.

4.3.3.1 Investigate the technology (Step 1)

Since this model will always be used to make decisions within a technology context, it is important to gain an understanding of the relevant technology and the choices that must be made regarding the technology. In the context of this study, this means investigating NoSQL technologies and families. Refer to Chapter 3 for more information on NoSQL families.

The technology focus of this study lies within the NoSQL technology area. Each of the NoSQL families represents a different data model with unique strengths and weaknesses.

4.3.3.2 Identify the comparison criteria (Step 2)

In Step 1, a good understanding of what the technology involves is gained. In Step 2, this knowledge is used to create a fixed set of criteria that are important to the specific technology. The development of the fixed set of criteria for NoSQL technologies is discussed in Chapter 5. The fixed set of criteria enables the NoSQL families and products to be uniformly compared.

The fixed set of criteria will ensure that only relevant information is used to compare NoSQL families with one another. Therefore, no unnecessary information can influence the decision-making process. This counters the *not enough meaning* category of biases. A fixed set of criteria will encourage investigation, which will counter the *memory effects* category of biases. Individuals will not need to rely on past experiences to compare the NoSQL families but will be able to base their decisions on investigated information. The criteria can also be adapted to a specified context (NoSQL), which ensures contextual information is used to compare the families. This will counter the *not enough meaning* category of biases.

4.3.3.3 Grade according to the criteria (Step 3)

Each option must be graded based on the extent to which it fulfills each of the criteria. This will ensure that the decision-maker considers all possible options and thereby remove some bias. In the context of this study, the options will be graded according to nine criteria. These criteria will be developed in Chapter 5, and grading will be discussed in Chapters 6 and 7. Chapter 6 will provide information pertaining to the application of the process model to a NoSQL technology context. Chapter 7 will apply the process model grading system in the context of instantiation meaning the performance of the four families will be graded.

Once the fixed set of criteria has been identified, grading can commence. Each NoSQL family is graded using the fixed set of criteria. The grades represent the performance level of a specific NoSQL family for each of the criteria. Examining all the grades of a NoSQL family can show its unique strengths and weaknesses. High precision values would not heavily influence the performance levels of the families. Within the context of this study, there is no need for a decimal value to depict the performance levels of the families since the decimal values may not change the meaning of the performance level. Therefore, high accuracy values are used to depict the performance level of each family.

Grading the families by using the criteria will ensure that IT practitioners investigate the NoSQL families by following systematic steps to provide values for all of the criteria. This counters the *act fast* and *memory effects* categories of biases.

Depending on the use case, certain criteria will be more important than others. Weighting the criteria will show the different levels of importance of the different criteria in the context of the specific use case.

4.3.3.4 Weight the criteria (Step 4)

Step 4 entails weighting the criteria to indicate the level of importance of each criterion within the context of a specific use case. Since each use case has specific requirements, the criteria are not all of equal importance. Therefore, it is crucial to allow IT practitioners to input weights into the model. In the context of this study, the weighting of criteria is done within the context of a NetFlow use case and is discussed in Chapters 6 and 8. Chapter 6 will provide information relating to the application of the process model in the context of the use case. Chapter 8 will apply the process model weighting system in the context of the NetFlow use case (instantiation).

A tool that will enable an IT practitioner to assign weights to the different criteria is required. A fixed amount of points is provided to the IT practitioner to assign to the various criteria. The tool is an essential element of the weighting process, as it prevents the IT practitioner from assigning equal weights to all the criteria. Therefore, the IT practitioner is forced to consider which of the criteria are more important and which are less important. Ensuring that proper weights are assigned to the criteria will increase the quality of the recommendation from the model.

The *average of values* measurement can influence the final weight values assigned to the criteria. An average of the weighted values received from respondents is used to derive a common weight value for each criterion. However, the derived value may not represent all the respondents' specific weight values. Therefore, the wrong requirements may be used to base decisions on. The weights of the criteria must represent the level of importance of each criterion within the specific use case. High precision values would not change the meaning of the final weights for each criterion. A decimal number would not change the meaning of the weight. Therefore, high accuracy values are adequate for the weight values within the model.

The model will provide weighted criteria by using relevant information to determine the importance value of each criterion within the use case. Using only relevant information will counter the *too much information* category of biases, because no unnecessary information will be able to influence the weightings of the criteria. Weighting criteria also forces the IT practitioner to investigate only contextual information regarding the importance level of each criterion. This counters the *not enough*

meaning and *memory effects* categories of biases. Focus is also placed on the importance level of each criterion. This forces the individual to investigate the importance of each criterion to ensure the specific use case requirements are represented.

4.3.3.5 Score the options (Step 5)

Once Step 4 has been completed, the options can be scored. The final score for each family is derived from a combination of the weights (Step 4) and grades (Step 3) of its criteria. The grades are used to indicate the unique strengths and weaknesses of each family, while the weight values represent the importance of each criterion within the context of the use case. Scoring the options is discussed in Chapters 6 and 8. Chapter 6 provides information relating to the process model, while Chapter 8 focuses on how the weights are applied during instantiation. In the context of this study, the weights represent the importance of the criteria within a NetFlow use case.

The accuracy and precision of values influence the scoring of the NoSQL families. When scoring, only high accuracy values are used meaning high precision values would not influence the meaning of the final score. Therefore, high accuracy values can be used to determine the final score of each option.

A calculation is used to determine the score for each criterion. A criterion's score is a combination of its weight and grade. Once all the criteria have been scored, the NoSQL families' final scores can be calculated. All the criterion scores of a NoSQL family are added together to obtain its final score. The final score is an indication of the appropriateness of the family for the particular use case. This calculation process is followed to obtain the final score of each of the NoSQL families. The final score of each family will be different, as each family has its own unique strengths and weaknesses. The next step is to make recommendations based on the final scores the NoSQL families.

4.3.3.6 Recommend an option (Step 6)

The last step of the framework is to provide a recommendation based on the final scores of the options. Completing Step 5 provides the final scores, which should differ from one another. The highest score indicates which option is most appropriate for the use case. The recommendation are based on a mathematical formula, however the recommendation is not perfect. Therefore, the recommendation reduces uncertainty but does not remove uncertainty.

In the context of this study, the NoSQL families are scored to find the one most appropriate to be used in the NetFlow use case. The final scores represent the ability of each NoSQL family to fulfil the specific NetFlow use case requirements. The higher the final score, the more a specific NoSQL family meets the requirements of the use case. In Step 6, the final scores of the NoSQL families are

displayed and a recommendation is made based on them. The family with the highest final score will perform the best within the context of the specific use case.

Figure 4.3 provides a graphical overview of the artefacts comprising the proposed framework. The weighetd decision model combines several constructs which include the list of options, the comparison criteria, the importance weightings, the performance grades, the final scores for each option and the 6-step process which can adapt the framework to specific technologies.

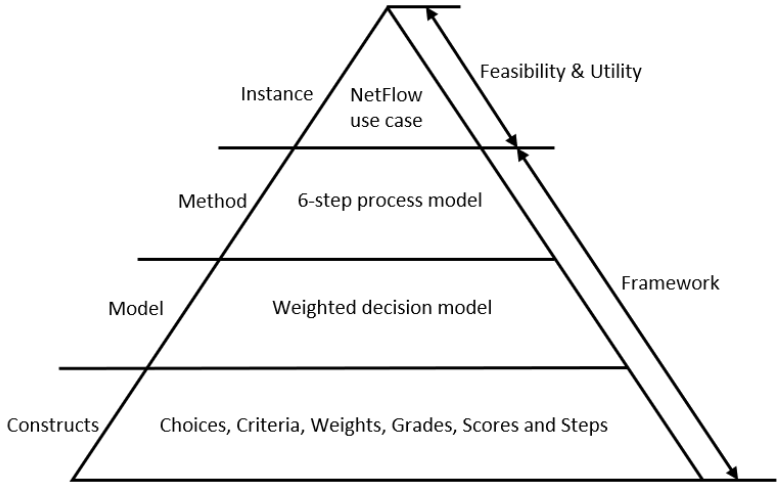

Figure 4.3: Overview of the framework.

4.4 Conclusion

In previous chapters, it was established that there is a need for a framework that can help IT practitioners make better decisions regarding technology. Therefore, this chapter proposed a framework that can assist with technology-based decisions.

Figure 4.3 provides a graphical overview of the artefacts that make up the proposed framework. The weighted decision model combines several constructs, including the list of options, the comparison criteria, the importance weightings, the performance grades, the final scores for each option, and the 6-step process that can be used to adapt the framework to specific technologies.

In Chapters 5 and 6, the weighted decision model within the proposed framework will be modified to enable it to be applied to the context of NoSQL technologies. The adjustments that will allow it to be used in a specific technology context rather than a general technology context will be made by using the 6-step process discussed in this chapter. Chapter 5 will investigate the development of the comparison criteria to ensure that a uniform comparison of the NoSQL families can be made. Chapter 6 will discuss how to implement various steps in the process model within the context of NoSQL technologies.

The next part of this study uses the proposed framework within a specific use case to demonstrate the feasibility and utility of the framework. This is indicated by the top layer of Figure 4.3. The framework is employed within a NetFlow use case context to assist IT practitioners in deciding which NoSQL family is most approproiate for storing Netflow data. The next chapter will develop a set of criteria that can be used to compare NoSQL families.

CHAPTER 5: CRITERIA DEVELOPMENT

In previous chapters, it was made clear that making decisions regarding NoSQL families is difficult. This motivated the need for a framework that IT practitioners can use to make better decisions regarding technologies. In response to this need, Chapter 4 proposed a framework to help counter the problems that can influence technology decision-making. The framework comprises several constructs that work together within a decision model. It also proposes a 6-step process to adapt the decision model for specific technologies to assist in decision-making regarding a specific technology.

Chapters 5 and 6 set out to customise the decision model to the context of this research study. This will enable it to be used to decide between NoSQL technologies. Step 1, investigating the technology, was completed in Chapter 3 of this dissertation. Chapter 5 focusses on Step 2, which is identifying comparison criteria. To assist IT practitioners in choosing between NoSQL families, a fixed set of criteria will be used to uniformly compare the families.

The chapter starts off by explaining the need for comparison criteria and provides an example of existing comparison criteria. Thereafter, additional criteria are added to the existing criteria to create to the full set of comparison criteria that will enable the NoSQL families to be uniformly compared.

5.1 Why develop comparison criteria?

Chapter 3 identified the four existing NoSQL families, each of which uses a specific data model. The families all have unique strengths and weaknesses that need to be compared so that an informed decision regarding which family to select for a specific use case can be made. Therefore, a uniform comparison must be made between the four families.

The criteria are based on certain aspects of NoSQL databases that can be used to uniformly compare the families. The aim of the criteria is to enable the framework to indicate the unique strengths and weaknesses of each family. The criteria will also assist in combatting some of the bias categories (section 2.5) by ensuring the IT practitioner considers all possible alternatives for the use case.

An example of a method that can be used to compare NoSQL families is the CAP theorem. The CAP theorem states that a NoSQL family can only contain two of the three CAP properties. The three CAP properties are consistency, availability, and partitioning (Brewer, 2000). The CAP theorem is an example of a fixed set of criteria in the comparison of NoSQL databases (Brewer, 2000, 2012). In the following sections, the CAP theorem is investigated and expanded on using additional criteria to enable a better uniform comparison of the NoSQL families.

5.2 The CAP theorem

As stated in section 3.1.1, most relational database systems are based on transactions to ensure the atomicity, consistency, isolation, and durability (ACID) of data management (Moniruzzaman & Hossain, 2013). However, storing large volumes of semi- and unstructured data may cause ACID-compliant relational databases to struggle. Therefore, to overcome the shortcomings of ACID, Brewer (2000) proposed the CAP theorem.

The CAP theorem describes a NoSQL database in terms of consistency, availability, and partition tolerance (Brewer, 2000, 2012). Firstly, high consistency refers to the ability of the system to always provide clients with the most up-to-date data (Brewer, 2012). Secondly, high availability refers to the ability of a system to ensure successful reads and writes most of the time (Brewer, 2000, 2012). High availability may also refer to the expectation that each operation will terminate successfully (Pokorny, 2013). Thirdly, high partition tolerance refers to the ability of a system to accept read and write requests even if network partitions are unavailable (Brewer, 2012; Pokorny, 2013). The three properties (CAP) should be balanced against one another when considering a database management system.

A NoSQL database can only possess two of the three desirable CAP properties (Brewer, 2000, 2012). Table 5.1 lists several NoSQL products and their corresponding CAP properties. There are a wide variety of products with different combinations of the CAP properties. This implies that there may be no single solution to all problems. Instead, there are many products which can be used to solve a variety of problems. The possible combinations of the CAP properties are AP (Availability-Partition), CP (Consistency-Partition), and AC (Availability-Consistency).

Table 5.1: Tabular format of CAP properties for popular NoSQL databases (Hu, Wen, Chua, & Li, 2014).

Data model	Technology	CAP option
Key-value	Dynamo	AP
	Voldemort	AP
	Redis	AP
Column-family	BigTable	CP
	Cassandra	AP
	HBase	CP
	Hypertable	AP
Document-based	SimpleDB	AP
	MongoDB	AP
	CouchDB	AP
Graph	PNUTS	AP

Figure 5.1 also indicates the CAP combinations and some of the products associated with each combination. The products indicated are only examples and not a complete list. The AP combination of properties means that full consistency is not a goal of the system. Higher availability and partition tolerance (AP) is the aim of most NoSQL systems (Moniruzzaman & Hossain, 2013). The CP

combination of properties means that high availability is not important to the system (Han et al., 2011). The AC combination of properties means that high partition tolerance is not a priority (Han et al., 2011). If high consistency is not part of the combination of properties of a NoSQL system, the system becomes basically available, soft-state, and eventually consistent (BASE) (Hecht & Jablonski, 2011; Sharma & Dave, 2012). Therefore, systems that has do not have a focus on providing high consistency will provide availability and partition tolerance. The NoSQL system may restrict the data model to enable better partitioning (Moniruzzaman & Hossain, 2013).

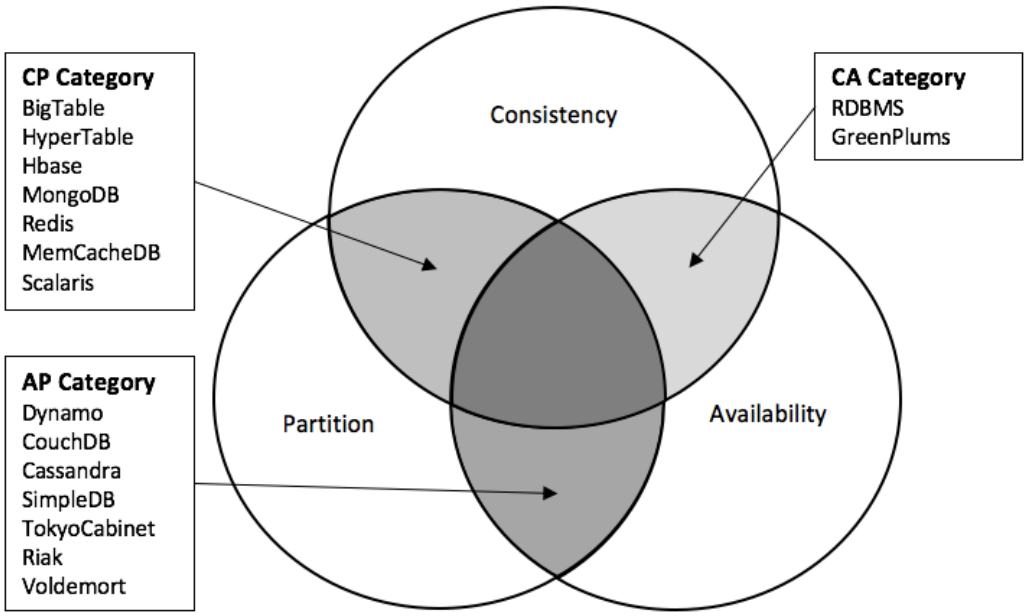

Figure 5.1: CAP theorem combined with NoSQL database products (Piplani, 2010).

The four NoSQL families have varying levels of abilities. A comparison must be made to help IT practitioners decide between the families. The CAP criteria provide a platform that can be used to compare the NoSQL families. However, this places the focus on only three criteria, while other factors are ignored. The CAP criteria are centred around the state of the data in storage. However, CAP does not consider how the data is processed to be stored later. Additional criteria will be able to provide a more holistic view that can be used to enable a better comparison between the NoSQL families. Therefore, additional criteria need to be identified to ensure a holistic and uniform comparison.

5.3 The fixed set of criteria

Throughout the literature review process, certain prominent criteria were identified by investigating data storage technologies. These criteria can be used to provide IT practitioners with a more informed and less biased view of the NoSQL families. Using a fixed set of criteria will ensure that the families are uniformly compared with one another. This will assist in mitigating the problems identified in Chapter 2. The identified criteria comprise of the CAP theorem (*consistency*, *availability*, and

partitioning) as well as some additional criteria. The additional criteria are *read and write performance*, *scalability*, *conceptual data structure*, *reliability*, and *learning curve*.

These criteria represent certain abilities and aspects of database technologies. The identified criteria cover the overall abilities of the NoSQL families. An explanation and justification of each criterion follows.

5.3.1 Consistency

Completing a write operation will insert a record into a database. If the database system has high consistency, all readers will immediately see the most up-to-date information (Brewer, 2000, 2012; Strauch et al., 2011). Therefore, consistency refers to the extent to which the system is in a consistent state after operations such as reading and writing have occurred (Chen, Mao & Liu, 2014). The level of consistency in a database system depends on the requirements of the use case. There are several levels of consistency, including strong, weak, and eventual consistency (Gilbert & Lynch, 2012; Vogels, 2009).

Strong consistency refers to a level of consistency at which any client accessing the data after an update to the data set will immediately see the most up-to-date version of the data (Lourenço, Cabral, Carreiro, Vieira & Bernardino, 2015b; Moniruzzaman & Hossain, 2013; Pokorny, 2013; Vogels, 2009). Weak consistency refers to a level of consistency at which accessing the data subsequent to an update does not guarantee that the most up-to-date version will be displayed. Certain conditions need to be met before the up-to-date data can be returned (Pokorny, 2013; Vogels, 2009). Eventual consistency is closely related to weak consistency. Eventual consistency means that the current data will become the most up-to-date data if no new updates are made to the dataset (Lourenço et al., 2015b; Vogels, 2009). The eventual consistency model has several variations, such as causal, read-your-writes, session, monotonic read, and monotonic write consistency, that can be used depending on the use case requirements (Vogels, 2009). The use case influences the level of consistency required to provide a specific service. An advantage of having weak consistency is that availability and scalability levels increase within the system (Brewer, 2012).

According to the ACID model, relational databases support full consistency most of the time (Pokorny, 2013). Strong consistency can affect database performance, as the data needs to be kept up to date constantly. Complex application logic is employed to detect and resolve any inconsistencies to constantly provide up-to-date information. However, the complex logic affects the database performance, as more time needs to be spent on consistency (Pokorny, 2013). If semi- or unstructured data is stored in a relational database, the performance will degrade.

The CAP theorem allows NoSQL databases to employ either strong or eventual consistency, depending on the grouping of CAP properties (Brewer, 2000, 2012). According to the CAP theorem, NoSQL databases can provide high performance and strong consistency (Han et al., 2011). The groupings of CAP properties influence the abilities of the database system. The possible grouping of properties for consistency are CP and AC (Brewer, 2000, 2012). These groupings mean that the NoSQL databases have flexible data models that can accommodate different needs. Relational databases provide full ACID support and focus on storing structured data. Storing semi- and unstructured data in a relational database can lead to much slower performance, as time is wasted on transactions.

Depending on the use case, there may or may not be a need for strong consistency. If the use case works with semi- or unstructured data, non-relational databases should be considered. High consistency is a requirement when working with transactional data, such as banking data. The data must always be up to date. However, high consistency may not be a requirement when working with decision-making data. An example of decision-making data is network monitoring data such as NetFlow data. NetFlow is an instrument used in the Cisco IOS software to monitor and characterise network operation (Cisco, 2012). NetFlow captures data between two hosts for a period of time. A set of old NetFlow data can be used to base decisions on, because the data does not have to be updated anymore. A set of NetFlow data for a past time period can be used to make decisions regarding the network, therefore it does not have to be updated.

5.3.2 Availability

Availability refers to the percentage of time a system operates correctly and is deemed as running (Domaschka, Hauser & Erb, 2014; Microsoft, 2005; Orend, 2010). Availability can also mean that continuous operation occurs even if a fault is present (Chen et al., 2014; Strauch et al., 2011). In other words, availability refers to the uninterrupted operation of the service (Gilbert & Lynch, 2012; Han et al., 2011). Availability is a guarantee that clients will receive at least one copy of the data even if nodes are down (Moniruzzaman & Hossain, 2013; Pokorny, 2013). An example of availability is a server that has a 95% uptime, meaning 5% of the total running time the server was offline. However, availability does not reflect the frequency of the interruptions that occurred during the 5% downtime.

NoSQL database systems can provide availability more easily than SQL database systems, because NoSQL favours availability over consistency (Lourenço, Abramova, Vieira, Cabral & Bernardino, 2015a). There is a trade-off between the consistency and availability properties of the CAP theorem. If high availability is present in a system, it will have lower levels of consistency. Some NoSQL products allow the trade-off to be managed by adjusting the levels of consistency and availability.

An example of a database that allows the management of the tradeoffs is *Dynamo* (DeCandia et al., 2007).

Nelubin and Engber (2013) conducted a study that tested NoSQL products and their failover characteristics. The database technologies tested were *Aerospike*, *Cassandra*, *Couchbase*, and *MongoDB*. The results of the study showed that *Aerospike* and *Cassandra* had the shortest downtime, while *MongoDB* had the least favourable downtime (Nelubin & Engber, 2013). Therefore, different NoSQL technologies provide varying levels of availability to solve different problems.

High availability is a requirement for systems that aim to spend a high percentage of time operating correctly. For example, an online instant messaging service requires the database system to be available at all times to ensure all messaging operations occur. However, high availability may not be a requirement in instances where the database will not lose value if the system goes down.

5.3.3 Partitioning

Partition tolerance refers to a system's ability to continue functioning even if some network partitions are unavailable (Pokorny, 2013; Strauch et al., 2011). It is the ability of a database system to cope with the addition or removal of nodes (Brewer, 2000, 2012; Moniruzzaman & Hossain, 2013) and must be considered in situations where partitioning is present. If the volume of data exceeds the capacity of a database, partitioning the data must be considered (Domaschka et al., 2014; Strauch et al., 2011).

A partition tolerant database system will forward read and write requests to available nodes instead of offline nodes (Gilbert & Lynch, 2012; Han et al., 2011). Once an offline node comes online, the node will receive its requests that were intended for it (Pokorny, 2013). The database system must ensure that the write operations finish only if the nodes have replicated their stored data onto other nodes.

Partitioning the database across other clusters is a solution to capacity and performance problems (Moniruzzaman & Hossain, 2013). Relational databases scale vertically to address these problems. Scaling vertically refers to upgrading hardware (Hecht & Jablonski, 2011). However, upgrading the hardware of a server can be expensive and does not result in a linear increase of performance within relational databases.

NoSQL databases scale horizontally to overcome capacity and performance limitations. Scaling horizontally means employing multiple machines to exponentially increase the capacity and performance of the database system (Hecht & Jablonski, 2011). NoSQL families differ in the way they partition data across multiple machines (Hecht & Jablonski, 2011). Some of the NoSQL families

have a type of key-oriented data model (Hecht & Jablonski, 2011) where the key is used to store, identify and sort data (Abramova, Bernardino & Furtado, 2014). When partitioning data within a NoSQL system, two key-based strategies are employed to distribute data sets.

The first partitioning strategy is *range-based partitioning*, which entails distributing data sets according to the range of their keys (Hecht & Jablonski, 2011). Splitting a key set into blocks allows a routing server to assign these blocks to various nodes in the cluster (Chen et al., 2014). Each node handles the performance and storage of its assigned block of keys (Sharma & Dave, 2012). Queries searching for a specific key are first sent to the routing server and then assigned to the appropriate node to allow efficient handling of queries (Hecht & Jablonski, 2011). Therefore, an advantage of this method is that the routing server handles the partition block allocations and load balancing (Chen et al., 2014). However, a disadvantage is that the availability of the entire cluster is dependent on the single routing server (Hecht & Jablonski, 2011). This means that a single point of failure exists. To counter this disadvantage, the routing server is often replicated to other machines.

The second partitioning strategy is *consistent hashing*, which allows for higher availability (Hecht & Jablonski, 2011; Karger et al., 1999). This strategy employs a shared nothing architecture with no single point of failure. A shared nothing architecture refers to a distributed-computing architecture where each node in the cluster is independent from other nodes (Stonebraker, 1986). Therefore, none of the nodes share resources such as memory and storage capacity (Stonebraker, 1986) Hash functions distribute the keys randomly and allow for quick calculation of a key's address within the cluster (Hecht & Jablonski, 2011). Consistent hashing does not require a load balancer as range-based partitioning does. However, the addition or removal of nodes may have a negative impact on the performance of the system. The performance may be negatively influenced, because the keys are randomly distributed throughout the cluster and the addresses of the keys need to be re-calculated with the addition or removal of nodes (Hecht & Jablonski, 2011).

A highly partition tolerant system will continue to function without being affected by the addition or removal of nodes. High partition tolerance is required in situations that need a high fault tolerance, such as LinkedIn accounts. The users need to be able to access their LinkedIn accounts to make be in contact at all times and not miss opportunities. However, low partition tolerance can be utilised in situations where no data needs to be partitioned. If there are no other nodes in the server, partition tolerance will not be a critical requirement.

The CAP criteria have now been thouroughly investigated. The following sections will discuss additional criteria that represent some other features and aspects of the NoSQL families. They have been included to enable a more holistic comparison between the families.

5.3.4 Read and write performance

Read and write performance refers to the performance output of a database and the time it takes to complete a function. The read and write requests are sent to the database to complete a function or a request from a client. The client expects that the database to respond quickly without any noticeable delay (Hecht & Jablonski, 2011).

NoSQL databases may be the solution to performance demands when storing and working with semi- and unstructured data. NoSQL database performance can be optimised for both reads and writes (Lourenço et al., 2015a). The read/write optimisation depends on the tools used for the retrieval, storage, and organisation of data. Write optimisation means that a higher level of performance is experienced with write functions than with read functions. However, several NoSQL databases are in-memory stores meaning they can be optimised for read or write performance (Lourenço et al., 2015a). A database's performance may differ considerably depending on its optimisation. Some cases may require more reading performance than writing performance or vice versa. Therefore, within the model, reading and writing performance are included as two separate criteria.

5.3.5 Scalability

Scalability refers to a system's ability to deal with increasing workloads (Orend, 2010). Scalability of a database represents the performance change that occurs with the addition or removal of nodes. The addition of improved hardware or nodes impacts the performance and capacity levels of a database system (Kuhlenkamp et al., 2014). In a scalable system, the performance and capacity increase is proportional to the amount of hardware added (Agrawal, El Abbadi, Das & Elmore, 2011). There are two methods according to which systems can be scaled when hardware resources are added.

The first method is to scale *vertically* or follow the scale-up approach. To scale up means to add resources to a single node inside a system (Agrawal et al., 2011; Kuhlenkamp et al., 2014). Adding improved processors to a single server and increasing the storage capacity of a single server are examples of scaling vertically (Agrawal et al., 2011). Scaling vertically improves the performance of a single node.

The second method is to scale *horizontally* or follow the scale-out approach (Microsoft, 2005; Moniruzzaman & Hossain, 2013). Adding more nodes to the system leads to horizontal scaling. The performance and capacity of the system scale linearly with the number of servers (Pokorny, 2013). Thus, the addition of new servers to the system leads to an increase in capacity and performance proportional to the number of servers.

Relational and non-relational databases scale their performance and storage capacities differently. Relational database management systems use a scale-up method and scale vertically (Microsoft, 2005; Padhy et al., 2011). The scale-up method entails upgrading the performance and capacity of the server through hardware upgrades. Upgrading the performance and capacity of the server increases the performance of the relational database system (Naheman & Wei, 2013). Therefore, one of the most popular methods of scaling a relational database is running the database on a more powerful server (Leavitt, 2010; Padhy et al., 2011).

The greatest drawback to scaling up is the financial burden that must be carried in exchange for increased performance. Scaling up does not guarantee that the increase in performance and capacity will be proportional to the amount of hardware employed (Agrawal et al., 2011). The scale-up method has limitations. The data must be distributed across multiple servers and relational databases may not function well with data partitioning (Leavitt, 2010). Thus, the performance gain may not be as great, while the cost of the scale-up method will increase by a large margin.

Non-relational (NoSQL) databases can improve performance and capacity levels through horizontal scaling (Moniruzzaman & Hossain, 2013). Scaling horizontally means evenly distributing the workload among the nodes in a cluster. This scaling method provides performance levels that are proportional to the number of servers employed (Naheman & Wei, 2013). Horizontal scaling can assist in providing superior performance and adequate levels of storage with which to address the requirements of unstructured data. Consequently, scalability may be a crucial requirement when dealing with large volumes of unstructured and heterogeneous data (Lourenço et al., 2015a).

There exist drawbacks to scaling out. Horizontal scaling also has performance drawbacks since the type of deployment will impact the performance (Audette, 2011). For example, a scaled-up website will provide better performance than the scaled-out website when accessing a local database rather than over the internet. The scaled-out system will provide better capacity while having a higher latency when reacting to requests (Audette, 2011). Additionally, scaling out allows the addition of more nodes to the cluster to increase the processing and storage capacity. The addition can lead to higher initial and operational costs when compared to scaling up as there are more servers running (Audette, 2011). Employing horizontal scaling can lead to much higher financial costs if powerful servers are used within the cluster. Therefore, cost can be a major drawback to horizontal scaling too.

A highly scalable database can increase its performance and capacity levels to accommodate an increase in workload. The particular instance determines which method of scalability is employed. There are two methods (vertical and horizontal scaling) that can be used to achieve high scalability

within databases. If data needs to be scaled to various other nodes in the cluster, horizontal scaling is best the approach to use. NoSQL databases commonly employ horizontal scaling to achieve the best performance and capacity levels possible. If the data does not need to be scaled to other nodes, vertical scaling may be the best approach to use. Relational databases commonly employ the vertical scaling method.

5.3.6 Conceptual data structure

A substantial amount of data is created daily. Businesses run constantly, and individuals have free reign over content creation (Naheman & Wei, 2013). The data takes several forms, for example text, images, audio, and video (Gandomi & Haider, 2015). There are three types of data, namely *structured*, *semi-structured*, and *unstructured* data. Structured data refers to data that is organised using a pre-defined structure (Gandomi & Haider, 2015). There are many restrictions placed on the structure of the data. Semi-structured data refers to data that is similar to structured data but has fewer restrictions placed on its structure (Chen et al., 2014; Gandomi & Haider, 2015). The data structure can change to a certain degree. Unstructured data refers to data with no pre-defined structure (Chen et al., 2014). This type of data does not conform to any of the restrictions placed on it by the data model (Gandomi & Haider, 2015). Thus, it is schema-free data. These three data types must be stored in appropriate databases.

There are two main types of database systems, namely relational and non-relational database systems. Relational database systems store data in a structured format (Moniruzzaman & Hossain, 2013). Unstructured or semi-structured data can be stored in a relational database only if it has been transformed into a structured format. Storing unstructured or semi-structured data in a relational database may cause performance penalties (Abadi, 2009). Non-relational database systems can store semi-structured and unstructured data without transforming the data (Chen et al., 2014). Therefore, no performance penalties will occur.

There is little to no restriction placed on the non-relational data model (Chen et al., 2014). NoSQL database systems are non-relational. As mentioned in Chapter 3, there are four NoSQL families and each family represents a specific data model that can store unstructured and semi-structured data. However, each data model is also equipped to store specific types of data.

Key-value stores are well suited for quick retrieval of values, such as user profile data or online shopping cart data (Moniruzzaman & Hossain, 2013). Graph stores are well suited for data that is heavily linked and relationship heavy (Moniruzzaman & Hossain, 2013). An example of such data is routing data that can be used for online maps as well as by location-based services and recommendation services (Hwang et al., 2015). Document-based stores are well suited for large

volumes of semi- or unstructured data (Moniruzzaman & Hossain, 2013). Examples of such data can be found in content management systems and applications for e-commerce. Column-family stores are well suited for heavily distributed data storage and large-scale data processing (Moniruzzaman & Hossain, 2013). Examples of data stored in column-family stores include online transaction data, user profile data, and content management system data (Hwang et al., 2015). All the data types mentioned above are semi-structured or unstructured in nature. These types of data have elements that can change structure and need to be stored in databases that can handle this.

Thus, data structure influences which NoSQL database system is chosen. Certain data types must be stored within a specific database system that employs a specific data model. If the data structure is the overriding factor in the decision, the criteria can be used to decide between products of the same conceptual data model.

5.3.7 Reliability

As stated above, availability represents the percentage of time a system is up and running. However, there is no indication of the frequency of incidents occurring which is where reliability is found. Reliability is an indication of how often an incident occurs (Domaschka et al., 2014). Therefore, high reliability is an indication of the database system's ability to operate without frequent failures occurring (Domaschka et al., 2014). The reliability criterion can influence the operation time of a database. If the database is highly reliable, the database will perform its function with a low probability of frequent failures occurring. Storing sensitive business data would require a highly reliable database system. If a database experiences frequent faults, it may stop functioning, which can result in value being lost. Thus, the level of reliability represents the level of tolerance against failures. A system is more reliable if it is fault tolerant (Microsoft, 2005). Therefore, fault tolerance refers to the ability of a database system to continue operating if a part of the system fails (Microsoft, 2005).

Relational databases may currently be the dominating force in databases due to their ACID properties (Lourenço et al., 2015a). ACID properties indicate that the reliability level of a relational database is high (Leavitt, 2010). NoSQL databases do not provide the degree of reliability that relational databases do, as NoSQL is not ACID compliant. If an IT practitioner wants NoSQL to be ACID compliant, additional programming is needed.

However, NoSQL databases can also have high levels of reliability. If NoSQL databases aim to be highly reliable, then two questions regarding their operation need to be answered (Domaschka et al., 2014). The first question is: *How does the database resolve concurrent writes to the same item*? (Domaschka et al., 2014; Lourenço et al., 2015b). The second question is: *What level of consistency*

is observed by clients? (Domaschka et al., 2014; Lourenço et al., 2015b). These questions represent how reliability is provided within a NoSQL database system. Consequently, the answers to these questions represent the level of reliability within a NoSQL database system. Therefore, IT practitioners should investigate NoSQL technologies that can answer both these questions well in order to find highly reliable NoSQL databases (Lourenço et al., 2015b).

A highly reliable database allows continuous functioning without failures for an extended period of time. Online shopping websites requires highly reliable databases because transactional data needs to be committed immediately and requires the database system to be functioning properly. Decision-making analyses of data may not require highly reliable databases because failures would not affect their value. Therefore, the level of reliability needed depends on the specific instance.

5.3.8 Learning curve

The learning curve refers to the time and effort needed to set up a database technology. This time spans from installation to the point where information has been captured. The learning curve can also be a good indication of the complexity of a specific database. If the learning curve is high, certain factors need to be considered. Firstly, there will be many prerequisites to attend to before setting up. Secondly, setting up the database will be complicated and time-consuming. Lastly, it will take time to learn the database commands and become fluent in its operation.

The above-mentioned factors are some of the reasons the learning curve of a database is important. The time needed to set up a database may indicate its level of complexity. Therefore, setting up a database should be done as quickly as possible in a way that will allow the database to remain stable. The database is the storage medium for the data and should be able to start capturing and storing data as quickly as possible. If it takes a long time to set up the database to start capturing data, then it is an indication of a large learning curve.

As stated above, the learning curve of a database is a good indication of the complexity level of setting up and using the database. This criterion is hard to measure, as all use cases are not the same. Currently, most database guides, research papers, and database books available are about relational databases. Relational databases are still the dominant technology in the data storage area, which is why they are focused on in study materials. However, since unstructured data is also on the rise, more research on other technologies should be done.

Non-relational (NoSQL) databases are currently very prevalent. Therefore, various books and guides about them are available. These books and guides can teach individuals about NoSQL in general as well as how to set up specific NoSQL products. The amount of research into non-relational data

storage technologies is on the rise. Numerous websites have product documentation to assist individuals in setting up and using NoSQL databases.

The learning curve may have an impact on the selection of database systems. The lower the learning curve, the easier it will be to employ and use a database system. The higher the learning curve, the more complex and time-consuming it will be to employ a database system. The learning curve may be an indication of the potential value to be gained from a database system. A high learning curve may indicate the level of functioning the database system can achieve. Therefore, depending on the needs of the instance, either a high or a low learning curve can be selected.

5.4 Conclusion

In previous chapters, it was established that a fixed set of criteria is needed to uniformly compare the NoSQL technologies with one another. This chapter set out to identify and explain the criteria that will be used to compare the NoSQL families. The identified criteria are *consistency*, *availability*, *partitioning*, *read and write performance*, *scalability*, *conceptual data structure*, *reliability*, and *learning curve.*

The goal of the fixed set of criteria is to uniformly compare the families to assist IT practitioners in deciding between them. The above criteria represent certain capabilities and aspects of NoSQL families that IT practitioners must consider. Table 5.2 lists the criteria and the research studies that informed their adoption.

This chapter completed Step 2 (identify the comparison criteria) of the 6-step process model that is being used to adapt the decision model to the context of NoSQL databases. The following chapter performs Step 3 (grade according to the criteria), Step 4 (weight the criteria), and Step 5 (score the options) of the 6-step process model to further customise the decision model to be used to make choices regarding NoSQL databases.

Table 5.1: Studies for each criterion.

Criteria	Criteria definition	Primary sources
Consistency	Ability to retrieve the most up-to-date information.	Brewer, 2000, 2012; Chen, Mao & Liu, 2014; Domaschka, Hauser & Erb, 2014; Gilbert & Lynch, 2012; Han, Haihong, Le & Du, 2011; Leavitt, 2010; Lourenço, Cabral, Carreiro, Vieira & Bernardino, 2015b; Moniruzzaman & Hossain, 2013; Pokorny, 2013; Strauch, Sites & Kriha, 2011
Availability	Percentage of time the system is operating correctly.	Brewer, 2000, 2012; Chen, Mao & Liu, 2014; DeCandia et al., 2007; Domaschka, Hauser & Erb, 2014; Gilbert & Lynch, 2012; Han, Haihong, Le & Du, 2011; Lourenço, Abramova, Vieira, Cabral & Bernardino, 2015a; Microsoft, 2005; Moniruzzaman & Hossain, 2013; Nelubin &Engber, 2013; Pokorny, 2013; Orend, 2010; Strauch, Sites & Kriha, 2011
Partitioning	Ability to cope with the addition or removal of nodes in a cluster.	Brewer, 2000, 2012; Chen, Mao & Liu, 2014; Domaschka, Hauser & Erb, 2014; Gilbert & Lynch, 2012; Han, Haihong, Le & Du, 2011; Hecht & Jablonski, 2011; Karger et al., 1999; Moniruzzaman & Hossain, 2013; Orend, 2010; Pokorny, 2013; Sharma & Dave, 2012; Strauch, Sites & Kriha, 2011
Read and write performance	Time needed to complete a read query from a database. Time to needed complete a write function to a database.	Győrödi et al., 2015a; Győrödi et al., 2015b; Han, Haihong, Le & Du, 2011; Hecht & Jablonski, 2011; Lourenço, Abramova, Vieira, Cabral & Bernardino, 2015a; Naheman & Wei, 2013
Scalability	Ability to deal with increasing workloads.	Agrawal et al., 2011; Kuhlenkamp, Klems & Röss, 2014; Leavitt, 2010; Lourenço, Abramova, Vieira, Cabral & Bernardino, 2015a; Microsoft, 2005; Moniruzzaman & Hossain, 2013; Naheman & Wei, 2013; Orend, 2010; Pokorny, 2013
Conceptual Data Structure	The data structure influences the technology used.	Abadi, 2009; Chen, Mao & Liu, 2014; Gandomi & Haider, 2015; Hwang, Lee, Lee & Park, 2015; Moniruzzaman & Hossain, 2013; Naheman & Wei, 2013
Reliability	Level of fault tolerance.	Domaschka, Hauser & Erb, 2014; Leavitt, 2010; Lourenço, Abramova, Vieira, Cabral & Bernardino, 2015a; Microsoft, 2005
Learning curve	Time and effort needed to learn how to use and set up the database and the complexity level of setting up and using the database.	

CHAPTER 6: DECISION-MAKING PROCESS

Chapter 4 proposed a framework to assist with decisions regarding technologies. The framework presented a weighted decision-making model aimed at technology-based decisions in general. However, a 6-step process model can be used by IT practitioners to adapt the weighted decision model for use in the context of a specific technology.

The research problem deals with decisions regarding NoSQL databases. Step 1, investigating the technology, has been completed as part of the background study and identified NoSQL as the technological context. Chapter 5 was dedicated to developing a set of criteria that can be used to uniformly compare NoSQL families.

Chapter 6 provides a more detailed account of Steps 3 to 5 of the 6-step process model. The focus of Chapter 6 is indicated in Figure 6.1 by the shaded area. The following sections consider each of the steps in turn.

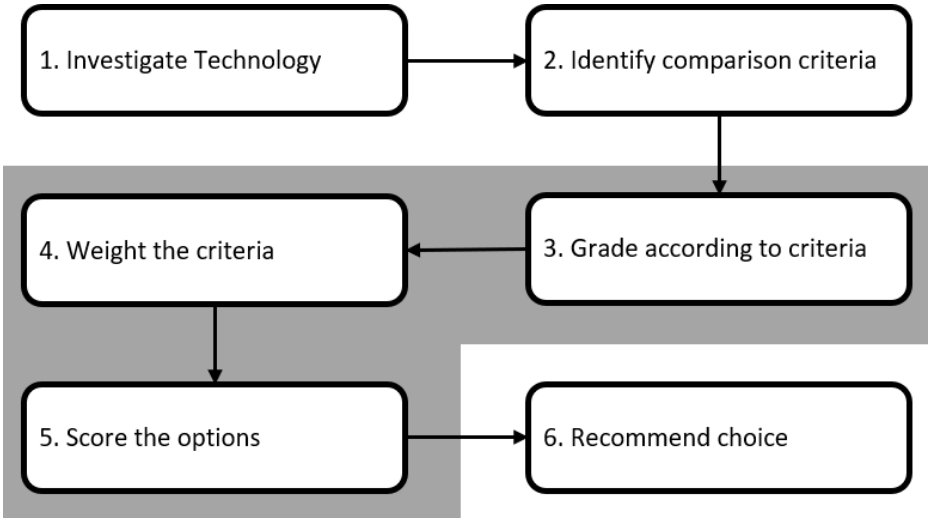

Figure 6.1: Focus of this chapter.

6.1 Grade according to the criteria (Step 3)

In Step 2, a fixed set of criteria that will be used to uniformly compare the technologies was identified. The uniform comparison aims to provide the IT practitioner with a holistic view of the options being compared. Such a holistic view assists with better decision-making. A specific approach is followed when grading the criteria to ensure appropriate grades are assigned and justified.

Step 3 is to assign grades to the criteria in the context of the specific technological options being investigated. Investigating each criterion will show the unique strengths and weaknesses of each technology option. Therefore, the assigned grade must represent the performance level of each

technology option. The result of completing this step is having an appropriate grade assigned to each criterion.

The model proposes that grades ranging from 0 to 10 be assigned. Each grade has a different meaning to indicate a specific level of performance. Table 6.1 contains the suggested grades with their associated meanings.

Table 6.1: Legend for grades assigned to the fixed set of criteria.

Grade	Meaning of grade
0	Criterion does not exist within the current technology.
1	Weakest performance. Many other more viable solutions available.
2	Weaker performance. May need another solution to increase performance.
3	Weak performance. May not be the best solution for this criterion.
4	Less than average performance. Can be improved upon.
5	Average performance. Much room for improvement.
6	Above average performance. Many improvements available.
7	Good performance. Alternative improvements available.
8	Very good performance. Few improvements available.
9	Great performance. Little room for improvement.
10	Most up-to-date/best solution is implemented. Cannot be improved upon, as it is currently the best solution available.

The grades can ensure a holistic view of the strengths and weaknesses of each technology. A combination of performance tests, document analyses, and other data collection methods leads to a grade out of 10. However, this study will rely on existing literature to assign grades when the framework is instantiated in Chapters 7 and 8. The next important aspect of the model to be discussed is the weight values.

6.2 Weight the criteria (Step 4)

Use cases represent real-world scenarios that IT practitioners work with. They have different requirements that IT practitioners must accommodate. Therefore, IT practitioners must be able to enter such information into the framework.

In Step 4, an IT practitioner enters weights that represent the importance level of each criterion as derived from a real-world use case. Each use case has unique requirements that must be represented in the decision-making model.

The criteria have varying levels of importance in a use case. If one criterion is more important than another, the weights must indicate this. The weights will influence the calculation of the final scores and thus also influence the recommendation. The advantage of assigning weights is that the relative importance of each criterion is considered and represented in the model.

6.2.1 The importance of weights

Table 6.2 provides an example of how the criteria weights for two use cases could differ. The values in this example were not empirically determined and are purely illustrative in nature.

The example illustrates that a use case based on SANReN (Use Case 1) and a use case based on an online shopping cart (Use Case 2) would assign different weights to the same criteria. Use Case 1 is a business related. SANReN (South African National Research Network) is capturing semi-structured Netflow data for future data analytics. This use case places focus on data analytics, writing and storing enormous amounts of semi-structured data, high scalability, high partition tolerance, eventual consistency, and some availability of the data. These properties are essential to the success of the use case.

Use Case 2 is an online shopping cart for an online retail store. The online retail store can serve tens of thousands of customers each day from thousands of servers, and each customer has their own cart. The goal is to provide customers with uninterrupted, highly available, scalable, reliable, and consistent access to the website while storing large amounts of semi- and unstructured data.

Table 6.2: Illustration of different weights based on use cases.

Criteria	Use Case 1	Use Case 2
Consistency	5	9
Availability	4	7
Partitioning	8	3
Read performance	3	5
Write performance	8	4
Scalability	7	7
Conceptual data structure	7	5
Reliability	6	8
Learning curve	2	2

The two use cases have different goals. Where Use Case 1 focuses on data analytics to provide value, Use Case 2 aims to provide uninterrupted access to a service. Therefore, the use cases' requirements also differ. This is why the weights of the criteria are different. The use case requirements determine the weights of the criteria. For Use Case 1, the read/write performance, partition tolerance, and scalability criteria are more important than consistency and availability, as the use case only requires eventual consistency and some availability. Therefore, higher weights will be assigned to the scalability, partition tolerance, and write performance criteria and lower weighting will be assigned to the consistency, availability, and conceptual data structure criteria.

In comparison, Use Case 2 requires high consistency, high availability, high scalability, high reliability, and good performance to provide clients with uninterrupted access to their data and

shopping carts. For Use Case 2, higher weights will be assigned to the consistency, availability, scalability, reliability, and conceptual data structure criteria. The requirements of the use case determine the weights, as some criteria are more crucial to the success of a project than others. The weights for the same criteria for these use cases will not be the same because the one use case requires these criteria more than the other.

In a business environment, IT practitioners should have adequate knowledge of the requirements of their use cases. The decision-maker compares the requirements of the use case to the criteria to identify the most important criteria for the success of the use case. The more important a criterion is to the success of the use case, the higher a weighting is assigned to that criterion. Of the criteria, the most important should be weighted the highest and the least important the lowest.

The following section deals with different techniques that can be used to determine the weights of the criteria.

6.2.2 Techniques used to determine the weights

There are several methods IT practitioners can use to obtain appropriate weight values to input into the model. Some of these methods are *interviews*, *focus groups*, and *questionnaires*. These methods will be discussed individually in the following section. How each method can be used by IT practitioners to enter appropriate weight values into the model will also be discussed.

Interviews refer to verbal interactions with other individuals (Kvale, 2008). Questions are posed and need to be answered by the respondents to provide information on topics. The goal of an interview is to construct knowledge from the interaction between the interviewer and the interviewee (Kvale, 2008). The interview may allow the interviewer to gain insight into the problem from the interviewee's perspective and thereby gain in-depth information regarding the problem. There are three different categories of interviews. The three categories are *structured*, *unstructured*, and *semi-structured* interviews (Qu & Dumay, 2011).

In the context of this study, *structured interviews* can be used to gather the weights of the criteria. The interviews must be conducted with experts that have extensive knowledge regarding the specific use case and its requirements. Before an interview can start, background information must be provided to the interviewee. The purpose of the interview must be disclosed as an investigation into the importance of the criteria in a specific context. The IT practitioner will ask the experts technical questions regarding the criteria and their importance to a specific use case. During an interview, special attention must be paid to the observed behaviour of the expert. The expert may indicate excitement or concern when asked a technical question. This can be an indication of the importance level of the criterion in question. The responses of the experts can be compared to indicate agreement

or disagreement, which can influence the weighting assigned to each of the criteria for a specific use case. The expert's justification of an answer may also be an indication of the appropriate weight value to be assigned to a criterion.

Interviews may be able to provide an indication of which criteria are the most important by obtaining the opinions of numerous experts. The experts' answers and behavioural reactions to the questions can be used to derive weight values that will indicate which criteria are more important and which are less important.

Focus groups refer to groups of individuals discussing a topic (Morgan, 1996; Stewart & Shamdasani, 2014). Focus groups (Morgan, 1996; Stewart & Shamdasani, 2014, pp. 7-8) are versatile because groups of individuals can discuss any topic and share ideas regarding any problem (Morgan, 1996; Stewart & Shamdasani, 2014). Interactions between individuals in such an environment may stimulate creative thinking regarding a topic. Focus groups are used in marketing research to address concerns regarding the design and service of products (Goldman & McDonald, 1987). In marketing research, the focus group can be used to obtain clients' perceptions of pricing, brands, and retail environments as well as their level of satisfaction with a product (Stewart & Shamdasani, 2014, pp. 7-8). Focus groups are user-friendly and can be analysed quickly (Stewart & Shamdasani, 2014). A focus group will lead to individuals supporting, contradicting, and extending the opinions of others and thereby provide new insights into a topic or problem (Stewart & Shamdasani, 2014, pp. 7-8). Without employing a focus group, such information might not be gained.

In the context of this study, a focus group can be used to derive the weight values of the criteria. A group of experts with knowledge regarding the specific use case must be gathered to discuss the importance of each criterion. The goal of the focus group is to have the experts interact with one another and determine the importance of each of the criteria. The experts will be able to express their opinions regarding the most important and least important criteria for the specific use case. The opinions of the experts may be the same. If so, the appropriate weight values can be derived easily. However, their opinions may also differ. If so, disagreements can be further discussed until a conclusion regarding the importance of the criteria is reached. The behavioural reactions of the experts can indicate agreement or disagreement. This can also be an indication of the importance of the criteria. Therefore, appropriate weight values can be assigned to the criteria by using a discussion group.

The value of a focus group lies in its ability to enable the IT practitioner to assign an agreed-on weight value to represent the importance of each criterion. The focus group enables in-depth discussions about the importance of the criteria to take place between experts. Therefore, the weight

values are derived from more than one opinion. If there is contention between the experts, in-depth discussions can result in the final weight value to be assigned. The opinions and reactions of the experts are assessable and can indicate the appropriate weight value for each criterion. Therefore, the IT practitioner can assign appropriate weight values to the criteria for a specific use case by using focus groups.

Questionnaires refer to questions posed to an individual to gain insight and retrieve unknown information regarding a topic (Gillham, 2011; Olivier, 2009). A questionnaire can have open-ended and close-ended questions (Gillham, 2011). Two types of instruments that can be used to ask close-ended questions are Likert scales and LPC scales (Olivier, 2009, p. 83). Likert scales can be used to measure the degree to which a statement applies to the respondent. When using a Likert scale, there must be a neutral point in the values. An LPC scale is comparable to a Likert scale. However, the respondent must provide a numerical value to indicate their preference to one of two alternatives.

Open-ended questions are not feasible for the purpose of the questionnaire in this study. In the context of this study, a close-ended questionnaire with LPC-like scales can enable experts to provide the appropriate weight value for each criterion. The LPC-like scales consist of the values 1 to 10. The value 1 represents the lowest weight value, and the value 10 represents the highest weight value that can be assigned. A total of 50 marks can be distributed between the criteria to indicate their various importance levels for the use case. Therefore, all criteria cannot have the same level of importance. The limit forces the experts to apply their minds to indicate the most appropriate importance level for each of the criteria. Thus, the limit may increase the quality of the recommendation in Step 6 of the framework.

What gives value to the questionnaire is that the experts are able to assign the weight values themselves. Therefore, the most appropriate weight values can be assigned to each criterion for the specific use case based on the experts' opinions. The responses of the experts can be compared with one another to indicate agreement and disagreement regarding the importance of each criterion. Outliers can be ignored to ensure a majority view. Thus, an appropriate weight value can be assigned to each criterion based on the majority opinion of numerous experts.

6.3 Score the options (Step 5)

The decision model compares different technologies with one another by grading each technology using a fixed set of criteria. The fixed set of criteria will ensure that a uniform comparison of the technologies can be made.

Table 6.3 depicts the decision model. The first column represents the set of *n* criteria (C_1 to C_n). The criteria represent the abilities of the technologies. Each criterion will have a weight assigned to it

based on the needs of the use case. The weights of the criteria are presented in the second column of Table 6.3 by W_1 to W_n. The different technologies (F_1 to F_m) will each be graded according to the criteria. Technology *k* (F_k) for criterion *i* (C_i) is assigned a grade (R_{ik}). The final score of a technology F_k (*Score(F_k)*) is equal to the sum of the weighted grades of all the criteria for that specific technology F_k.

The goal of the decision model is to assist an IT practitioner in making an informed decision. The decision model does so by comparing the criteria. The grades reflect the strengths and weaknesses of each technology. A justification of each grade shows the reasoning behind assigning that grade to a specific criterion. Weights assigned to the criteria recognise that not all criteria are equal, but that they must be considered in the context of a specific use case.

The decision model does not remove uncertainty completely. However, it aims to remove a degree of uncertainty and give direction to the decision process while combatting technology decision-making biases.

Table 6.3: The weighted decision model.

Criteria	Weight	F_1	F_2	…	F_k	…	F_m
C_1	W_1	R_{11}	R_{12}	…	R_{1k}	…	R_{1m}
…							
C_i	W_i	R_{i1}	R_{i2}	…	R_{ik}	…	R_{im}
…							
C_n	W_n	R_{n1}	R_{n2}	…	R_{nk}	…	R_{nm}

$$Score(F_k) = \sum_{i=1}^{n} W_i . R_{ik}$$

6.4 Conclusion

This chapter discussed Steps 3, 4, and 5 of the 6-step process proposed in the framework. These steps were grading the options according to the criteria (Step 3), weighting the criteria (Step 4), and scoring the options (Step 5). Each of these steps plays a critical role in the framework, which is used to make a recommendation.

Firstly, the grading of criteria aims to reflect the unique strengths and weaknesses of the NoSQL technologies. Secondly, the weights assigned to the criteria represent the requirements of the use case. These requirements are obtained through a questionnaire that IT practitioners must complete. The fixed set of criteria enables a uniform comparison of the technologies to be made. Lastly, the final scoring uses a method that combines the grades and weights of the criteria to derive a final score for each technology. The final score is used to make a recommendation regarding which technology to choose.

The last part of March and Smith's (1995) design science framework is instantiation. During instantiation, the model is placed within a specific instance to demonstrate its utility and feasibility. The next chapter discusses the instantiation of the model to demonstrate its use within the context of a specific use case.

PART C

INSTANTIATION

CHAPTER 7: GRADING THE NOSQL FAMILIES

Chapters 4 to 6 proposed a framework that can assist IT practitioners in making better decisions regarding technology. The framework included a 6-step process that can assist IT practitioners in adapting the framework to specific technologies. Chapters 7 and 8 will demonstrate the feasibility and utility of the proposed framework.

In Part B, the framework was proposed and adapted to assist with decisions regarding NoSQL technologies. Chapter 7 rates each of the four NoSQL families in terms of the criteria developed in Chapter 5. Using a fixed set of criteria ensures that a uniform comparison of the families can be made. Therefore, it is possible to have a holistic view of the technologies. Chapter 8 will weight the various criteria within the context of a specific use case concerning NetFlow data.

This chapter starts by discussing the grading step in the context of the case study. A product representative of each family is identified to be graded. Thereafter, each of the four families, represented by their respective products, is assigned performance grades for the criteria. Each family is investigated and graded individually to ensure a holistic view of their unique strengths and weaknesses.

7.1 Grading NoSQL families (Step 3)

Each use case has unique requirements that must be met for the project to be a success. Some requirements are more important for the success of the project than others. The use case used to demonstrate the use of the framework is found in the NoSQL environment. To make a decision regarding NoSQL, the four NoSQL families need to be compared to depict their unique strengths and weaknesses.

Grading takes place only once per type of comparison. In the context of this research study, grading will take place once in the context of NoSQL. Thereafter, the grades can be used for many use cases that deal with the selection of an appropriate NoSQL family. Grading can also be done once for a specific set of NoSQL products. Thereafter, the model can be used for many use cases that require a selection to be made from the same set of products. The utility of the framework is thus not limited to one specific use case.

The fixed set of criteria (Chapter 5) allows the families to be uniformly compared while combatting certain decision-making biases. Each family has a popular database product that can be used to represent it (DB-Engines, n.d.; ITBusinessEdge, n.d.; Mayo, 2016). The column-family stores are represented by *HBase*. *MongoDB* represents the document-based stores. *Neo4j* represents the graph stores, and the key-value stores are represented by *Redis*. Each of these database products are well

researched and were created by large organisations that will ensure that the development of the database technologies continues (HBase, 2007; MongoDB, 2008; Neo4j, 2017; Redis, 2017). They are well-developed, popular databases that are used in many organisations and have proven to be good benchmarks throughout the NoSQL environment (DB-Engines, n.d.; ITBusinessEdge, n.d.; Mayo, 2016). As a result, the four most popular database products are used to represent each of the NoSQL families. Table 7.1 gives a summary of the databases that represent the various families.

Table 7.1 Summary of choices.

Family	Represented by
Column-family stores	HBase
Document-based stores	MongoDB
Graph stores	Neo4j
Key-value stores	Redis

The following sections aim to explain each of the NoSQL families and assign grades to the criteria. A grade out of 10 will be assigned to each criterion for each of the NoSQL families. To motivate why these grades were assigned, the investigations into the families will be discussed. Column-family stores are discussed first. This is followed by discussions on the investigations into document-based stores and graph stores. Key-value stores are the last to be discussed.

7.2 Column-family stores (*HBase*)

HBase is an open source NoSQL database implementation based on Google's *Bigtable* data store (Cattell, 2011). *HBase* employs the column-family data model, which stores data in rows and columns (Naheman & Wei, 2013). A row, identified by a unique row key, can consist of multiple columns (George, 2011). Rows and columns belong to a specific table and many tables can exist. Each column contains a different version of the data and a different value is assigned to each cell inside the column (George, 2011). For example, a column that contains the home address of a user is created. The user later changes home address, and a new entry is made for that user. The old data is kept, and the new data is entered and linked to the same user. *HBase* adds a timestamp to keep track of these different versions of data. Each column value and timestamp combination is referred to as a cell (George, 2011). Figure 7.1 is a graphical representation of the data model in *HBase*.

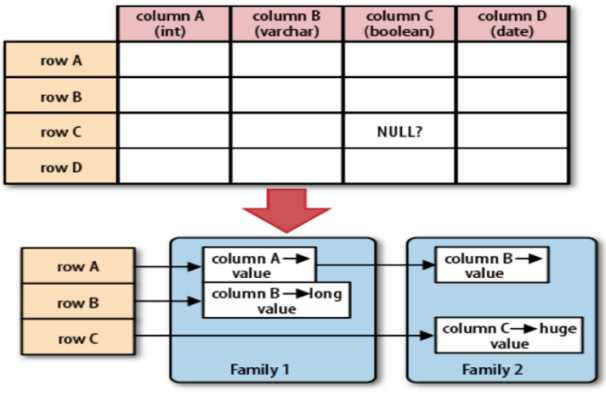

Figure 7.1: Representation of how *HBase*'s data model works (George, 2011).

Column families consist of groups of columns that are stored in the same file, known as an HFile, which is defined when the table is created (Dimiduk, Khurana, Ryan & Stack, 2013). There is no limit on the number of columns in a family or on the length of a stored value (George, 2011). The column-family data model stores data as a multidimensional sorted map and is accessed through a row key, column key, and a timestamp (George, 2011). To retrieve data, a client requires the family name, table name, row key, column key, and timestamp (Du Toit, 2016).

The investigation of the criteria for each NoSQL family starts with the consistency criterion.

7.2.1 Consistency

A write operation from a client to a database will insert or update records in the dataset. If another client reads the database contents and it is displayed the updated record immediately, then the consistency of the database is high (Brewer, 2000, 2012).

A study done by Hecht and Jablonski (2011) compared the four NoSQL families by investigating certain capabilities of these databases. Their study indicates that *HBase* can provide high consistency (Hecht & Jablonski, 2011) and is supported by the work of Dimiduk, Khurana, Ryan and Stack (2013), which also indicates that *HBase* can provide high consistency. High consistency means that clients can see the most up-to-date information immediately once it is written to the database. *HBase* operates in a multi-node cluster environment instead of a single machine. Therefore, *HBase* employs data replication to provide data consistency.

Replication refers to copying data between multiple *HBase* deployments. A log, known as the HLog, is created to keep track of the replications (George, 2011). Keeping track of the replications can ensure that consistency is provided with less effort. An *HBase* cluster can consist of several RegionServers with multiple regions, which refer to adjacent ranges of rows that are stored together (HBase, 2007). Each RegionServer can participate in the replication process to copy its data to other RegionServers. *HBase* employs Master/Slave replication to replicate the data between the different RegionServers (George, 2011).

The Master/Slave replication technique enables the data to be spread across nodes in clusters. There are two distinct roles that are assigned to the nodes in a cluster, known as a Master role and a Slave role (Gu, Wang, Shen, Ji & Wang, 2015). Only one Master role can be assigned at a time, while the rest of the nodes are assigned Slave roles (Gu, Wang, Shen, Ji & Wang, 2015). A Master node can replicate its dataset to any number of Slave nodes (George, 2011). Figure 7.2 shows an example of the Master/Slave process.

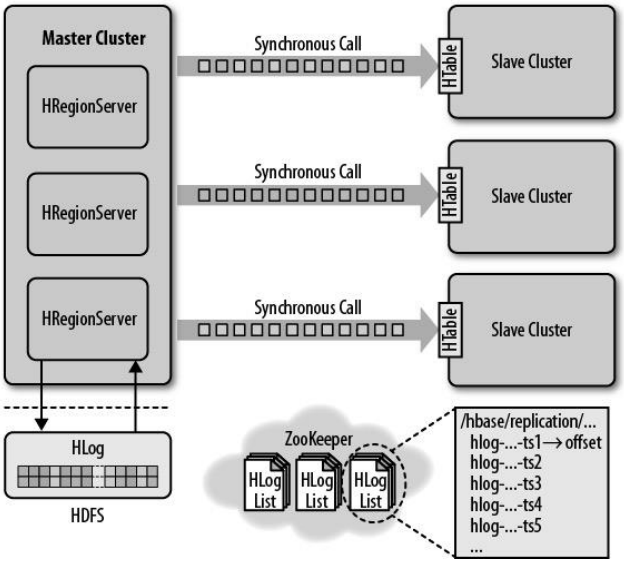

Figure 7.2: Master/Slave replication process (George, 2011).

HBase replication is done asynchronously, which means the data is written to the Master node and eventually to the Slave nodes (George, 2011). This refers to eventual consistency.

The basis of *HBase* replication is the HLogs from each RegionServer. These HLogs must be stored in a file system, such as the Hadoop file system, to ensure that replication to Slave clusters can occur (George, 2011). The RegionServer reads from the oldest log file to assist with the replication process. Therefore, the Master node will attempt to balance the stream of replication on Slave clusters by relying on randomisation (George, 2011).

Master/Slave replication is not without faults. If write operations are implemented on the Master node, the Slave nodes will forward the synchronise data command asynchronously to the Master node to update the Slave nodes' data (Gu, Wang, Shen, Ji & Wang, 2015). Read operations implemented on the Master node provides high consistency. However, read operations on Slaves provide only eventual consistency. Master/Slave replication does not provide automatic failover. Therefore, if the Master goes down, an election among the Slaves must occur to select a new Master node (George, 2011). The elected Slave must restart to change its role, which means there may be downtime. Also, if the number of write requests exceeds the capacity of the server, bottlenecks in performance can occur (Tauro, Aravindh & Shreeharsha, 2012).

As a result, *HBase* scores a grade of 7 for consistency of data. The rating of 7 justifies the use of Master/Slave in *HBase* to ensure consistency of the datasets. High consistency can be configured through Master/Slave in *HBase* to ensure data is consistent across the nodes. However, the drawback to this method is that when a Master node experiences a fault, a new Master node must be elected, and the election process may lead to downtime. Master/Slave replication can be used to provide high

consistency if implemented correctly. There are alternative options, such as sharding and the dynamo model, that can provide full consistency most or all the time. These options attempt to address the issues regarding the Master/Slave replication model for consistency of data.

7.2.2 Availability

Availability refers to the percentage of time that a system is operating correctly (Orend, 2010). A highly available database system aims to be available for client queries as long as possible before experiencing a fault.

Per the CAP theorem (Brewer, 2000, 2012), the combination of attributes for *HBase* is CP, which implies that there is a focus on providing data consistency and partition tolerance, while a degree of availability is sacrificed (Brewer, 2000, 2012; Cai, Huang, Chen & Zheng, 2013). Availability within *HBase* also refers to the ability of the system to handle node failures within the cluster (Dimiduk et al., 2013).

The *HBase* cluster consists of many nodes. Each node is referred to as a RegionServer. Each RegionServer has several regions that consist of adjacent ranges of rows stored together. Each RegionServer can serve multiple regions, while each region can only be served by one RegionServer (George, 2011). Therefore, *HBase* provides availability through the combination of Master/Slave replication and the RegionServers.

Master/Slave replication replicates the data to different RegionServers within the *HBase* cluster, thereby assisting the system in providing availability (Dimiduk et al., 2013; George, 2011). A RegionServer has access to the data of other RegionServers (Dimiduk et al., 2013; George, 2011). If a RegionServer experiences a failure, the data it was serving must be attended to by another RegionServer to ensure availability of service to a client. Therefore, *HBase* can still be available by enabling other RegionServers to attend to the faulty region of data (Dimiduk et al., 2013; George, 2011).

A drawback to this method is that when too many RegionServers are down, performance bottlenecks will occur, because the current RegionServers cannot attend to all regions. Another drawback is if the Master server or ZooKeeper is separated from the cluster, the Slave servers cannot function on their own (Figure 7.3) (Dimiduk et al., 2013; George, 2011). A solution to these drawbacks is defensive deployment schemes that can ensure higher availability (Dimiduk et al., 2013).

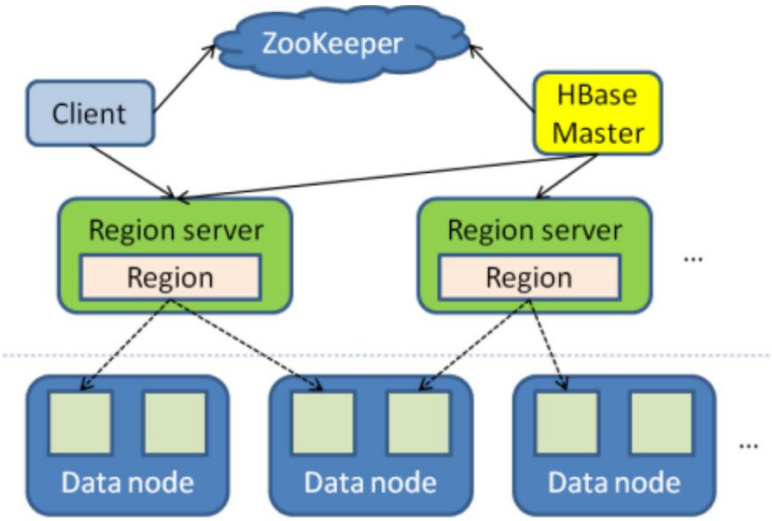

Figure 7.3: The *HBase* architecture (Gao, Nachankar & Qiu, 2011).

The type and amount of failures that *HBase* can handle is an indication of how strong its availability is. Lourenço et al. (2015a) investigated the availability guarantees of *HBase* and other database systems. Their research graded the database systems according to the level of performance the systems can provide. *HBase*, received a rating of "-" for the availability performance, implying that it may not provide the best levels of availability. Cai, Huang, Chen and Zheng (2013) agree that this database loses some availability, as its focus is on data consistency and partition tolerance. As a result, *HBase* receives a grade of 5 for the availability criterion.

A grade of 5 means that column-family stores is not be the strongest NoSQL family in terms of availability guarantees. However, they can be set up to provide higher availability through defensive deployments at the expense of other performance areas. A way to achieve higher availability is to configure more backup Master servers, which can mitigate the downtime of the election process.

7.2.3 Partitioning

Large volumes of data and a large number of read or write requests can lead to the capacity of a server being exceeded. Therefore, partitioning data to other servers may need to be considered. The data models of NoSQL databases are mostly key-oriented, meaning that partitioning is based on keys (Hecht & Jablonski, 2011).

There are two strategies that can be followed when implementing partitioning. The first strategy is to distribute datasets by the range of their keys. This is known as range-based partitioning (Hecht & Jablonski, 2011). A routing server is responsible for splitting the keysets into blocks, which are allocated to different nodes (Hecht & Jablonski, 2011). Once the allocation of blocks is completed, each node is responsible for request handling and storage of its specific keys (Hecht & Jablonski, 2011). To search for a specific key, the client should retrieve the partition table from the routing

server. A strength of range-based partitioning is that it can efficiently handle range queries, as it is highly probable that neighbouring keys are stored on the same server (Hecht & Jablonski, 2011). However, a weakness of this strategy is that the availability of the entire cluster depends on the fault tolerance of the routing server (Cai, Huang, Chen & Zheng, 2013).

The second strategy is consistent hashing, which provides a simpler cluster layout to counter the weaknesses of range-based partitioning by having no single point of failure (Hecht & Jablonski, 2011; Karger et al., 1999; Hecht & Jablonski, 2011). Keys are distributed using hash functions. Each server is responsible for a hash region. Therefore, the address of a key can be calculated quickly (Dimiduk et al., 2013; Hecht & Jablonski, 2011). The addition or removal of nodes affects a small portion of the entire cluster. However, the architecture and random distribution of keys lead to performance drawbacks, such as more processing time being spent on the calculation of the address of keys. (Hecht & Jablonski, 2011).

HBase implements range-based partitioning to partition its data and provide good performance (Nishimura, Das, Agrawal & El Abbadi, 2011). An increase in range query performance can occur if columns of the same family are stored on the same server (Hecht & Jablonski, 2011). The column-family data model can be partitioned efficiently, meaning these databases are more than adequate for large datasets (Hecht & Jablonski, 2011).

Hecht and Jablonski (2011) gave *HBase* a positive rating for range-based partitioning and a negative rating for consistent hashing. Their ratings mean that this database provides good performance with its range-based queries as a result of storing neighbouring keys next to one another. The database also hides the information that there are partitions present from the client application. Therefore, queries are less complex to perform (Dimiduk et al., 2013).

Given the above information, partitioning and partition tolerance within column-family stores receive a grade of 7. The grade of 7 means that *HBase* can provide good levels of partition tolerance while performing well where partitioning is concerned. However, the range-based partitioning strategy has its drawbacks. As stated above, the range-based partitioning strategy can provide good range query performance. However, the availability of *HBase* depends on the single routing server, which means there is a single point of failure that could result in downtime.

7.2.4 Read and write performance

A part of Du Toit's (2016) study evaluated the reading and writing capabilities of different NoSQL databases. The author inserted different amounts of records and recorded the time in milliseconds that the database took to execute the operations.

Du Toit (2016) executed a bulk write operation with various amounts of records within *HBase*. The author's research found that the average time used to insert a record within *HBase* increased as the amount of records increased (Du Toit, 2016). At 100 records, the time per record was 2.5 milliseconds, whereas at 20 000 records, the average time was 7.6 milliseconds per record (Du Toit, 2016). Du Toit (2016) found that the addition of data nodes leads to improvements in data writing performance. At 10 000 and 20 000 records, the addition of a fourth node led to an increase in performance greater than that of the other three nodes. This can be seen in Figure 7.4 (Du Toit, 2016).

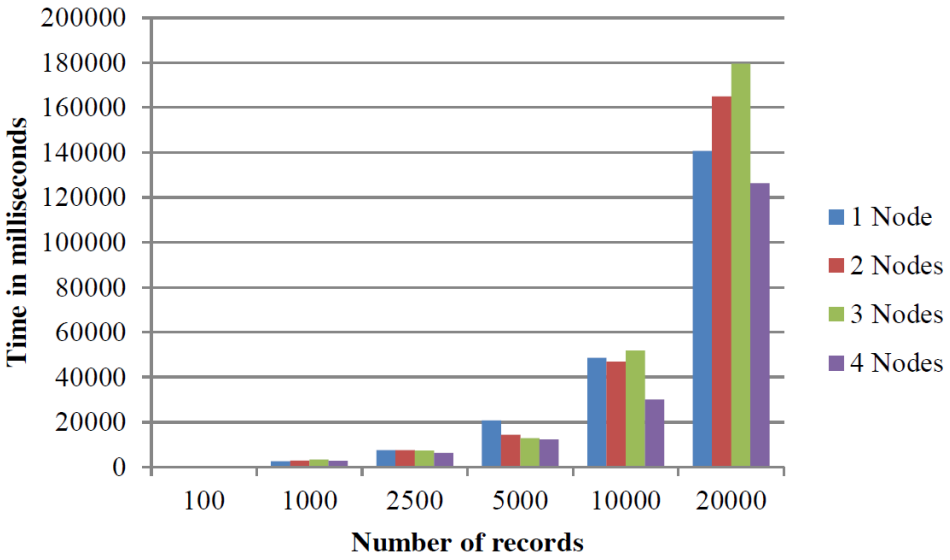

Figure 7.4: *HBase* data inserts average over four nodes (Du Toit, 2016).

Du Toit (2016) also tested the reading performance of *HBase*. The author executed read queries while recording the time needed to complete the queries. *HBase* reads single records in an average time of 3.7 milliseconds. Reading records in batches led to faster reading times ranging from 1.8 milliseconds (100 records) to 2.2 milliseconds (2500 records) (Du Toit, 2016). The author found that sets larger than 4000 records caused communication timeouts between the nodes (Du Toit, 2016). Therefore, to do a bulk retrieve query, a client has to create a list of get objects, which adds more processing time (Du Toit, 2016). His research could only produce results for read queries of up to 2500 records. Table 7.2 is a consolidated list of the reading performance for *HBase* within Du Toit's (2016) study.

Table 7.2: *HBase* data read average over four nodes (Du Toit, 2016)

Records	**1**	**100**	**1000**	**2500**
Duration in ms (average)	3.7	189.8	1614.8	5574.1
Latency per record in ms	3.7	1.898	1.6148	2.22964
Records per second	270.27	526.87	619.272	448.503

Write operations could handle considerably more records read operations. The considerable difference in performance between writing and reading may indicate that *HBase* is write optimised.

Lourenço et al. (2015a) compared NoSQL databases using a set of criteria by assigning a rating to represent the performance of the databases within each criterion. Lourenço et al. (2015a) compared the read and write performances of NoSQL databases, including *HBase*. In their rating system, *HBase* received a "+" for write performance and a "-" for read performance. These ratings imply that *HBase* is more oriented towards write performance than read performance.

A study done by Khetrapal and Ganesh (2006) examined *HBase*'s read and write performance. The tests included sequential and random read and write operations. The results (Table 7.3) showed that the sequential reads achieved a rate of 310 reads per second. The sequential writes achieved a rate of 1600 writes per second. The random reads achieved a rate of 290 reads per second, while the random writes achieved a rate of 1550 writes per second (Khetrapal & Ganesh, 2006). There is a large gap between the performance levels of reading and writing for *HBase*. Therefore, the results of the studies done by Khetrapal and Ganesh (2006), Du Toit (2016), and Lourenço et al. (2015a) agree that *HBase* is write optimised.

Table 7.3: Read and write speeds of *HBase* (Khetrapal & Ganesh, 2006)

Operation	**Rate**
Sequential reads	310 reads per second
Sequential writes	1600 writes per second
Random reads	290 reads per second
Random writes	1550 writes per second

Naheman and Wei (2013) attempted to inspect the relation of read and write performance to the number of column families in *HBase*. Their results show that *HBase* supports multiple column families that can store large volumes of different types of data (Naheman & Wei, 2013). Their study indicates that write performance is superior to read performance. This can be seen in Table 7.4. There is a significant difference in performance values when a higher number of records is reached, and the writing throughput is much higher than the reading throughput. Therefore, their study results suggest that *HBase* is write optimised and agrees with the previously mentioned studies.

Table 7.4: Comparison between read and write performance (Naheman & Wei, 2013).

	1 Region server					**8 Region servers**				
Experiment	1	10	100	500	1000	1	10	100	500	1000
Writes/sec	15	159	330	427	Timeout	3	35	160	384	Timeout
Reads/sec	10	93	142	128	Timeout	112	129	121	128	Timeout

A study done by Cooper et al. (2010) investigated the performance of some databases, including *HBase*. The authors performed several experiments, including reading and writing to the database. Their experiment results (Table 7.5) showed that the write performance of *HBase* is superior to its

read performance (Cooper et al., 2010). There is a noticeable difference between these two abilities of *HBase*, which implies that it is better able to handle write operations than read operations.

Table 7.5: Read or write optimisation of database technologies (Cooper et al., 2010).

System	Read or write optimisation
PNUTS	Read
Bigtable	Write
HBase	Write
Cassandra	Write
Sharded MySQL	Read

Literature provides sufficient evidence that *HBase* is optimised for write operations. There is a significant gap between its performance levels when writing and reading, which was identified in several studies above (Cooper et al., 2010; Du Toit, 2016; Khetrapal & Ganesh, 2006; Lourenço et al., 2015a; Naheman & Wei, 2013). These results may imply that *HBase* is a good option to consider for use cases that require high writing performance and average reading performance.

By taking all the above-mentioned points are into account, a grade of 8 is assigned to column-family stores for write performance and a grade of 4 is assigned for read performance. These ratings mean that this NoSQL family can provide very good writing performance and below-average reading performance. If a use case is write heavy and moderate on reads, then *HBase* may be an appropriate choice. Table 7.6 summarises the conclusions of the studies mentioned above.

Table 7.6: Comparison of read and write performance of different studies conducted.

Study	Read performance	Write performance	Optimisation (read or write)
Lourenço et al. (2015a)	Weak	Strong	Write
Khetrapal & Ganesh (2006)	Weak	Strong	Write
Naheman & Wei (2013)	Weak	Strong	Write
Cooper et al. (2010)	Weak	Strong	Write
Du Toit (2016)	Weak	Strong	Write

7.2.5 Scalability

Scalability refers to the system's ability to deal with increasing workloads (Orend, 2010). Column-family stores provide high scalability through partitioning data across multiple servers by splitting rows and columns (Cattell, 2011). Splitting rows and columns is done through sharding primary keys. Each database node in the cluster will store a shard and the range of data connected to that shard (Cattell, 2011).

A region is the basic unit of scalability in *HBase* (George, 2011; Dimiduk et al., 2013). In situations where a region's size is too large, the system will split the region into two or more regions to accommodate the size (George, 2011). Regions can also merge if they are small in size to reduce the

storage space and number of regions used (George, 2011; Dimiduk et al., 2013). When creating a table, there is only one region for that table. Once data is added to the table, a monitor checks if the table's size exceeds a configured maximum size. If the maximum size is exceeded, the region is split in two at the middle key, creating roughly two halves (George, 2011).

Each region is attended to by one RegionServer, which can serve many regions at any time. The splitting and serving of regions may be seen as autosharding (George, 2011). Autosharding regions enables rapid fault recovery if a server goes down. The regions can also be moved between the servers to assist with the load balancing of servers (George, 2011). Splitting the regions is fast because the split regions read from the original storage files (George, 2011, pp. 21-22).

HBase updates data on an atomic per row basis, which means that when applying an update to a row, that row is locked for the update period (George, 2011). The other clients that read or write to the same row will read a consistent last update or wait until they can update that row. That row cannot receive other updates until the current update is applied, meaning that if multiple clients try to update the same row at the same time, contention may occur (George, 2011, p. 75).

To deploy a fully distributed cluster for *HBase*, the Hadoop Distributed File System (HDFS) is required. The HDFS is the default file system for a distributed *HBase* cluster, since it has features that *HBase* requires to be deployed in a distributed environment. The HDFS has built-in fault tolerance, scalability, and replication to work with *HBase* and store data reliably (George, 2011, p. 54).

Lourenço et al. (2015a, 2015b) rated some capabilities of different NoSQL families that are represented by their respective databases. In their rating system, *HBase* received a "+" rating for scalability. A "+" rating is assigned if a database is geared for that specific property (Lourenço et al., 2015a, 2015b). Therefore, such a rating means that *HBase* is optimised for scalability.

A grade of 7 represents the scalability performance within column-family stores. Autosharding enables *HBase* to achieve high scalability, since the regions in *HBase* allow fast recovery of function if a server goes down. Autosharding also enables load balancing to balance the performance load of servers. The dataset can be split via the rows or the columns. Both row and column partitions can be used at the same time in the same table. Therefore, there are various ways to achieve high scalability, meaning high scalability is possible within column-family stores.

Another reason for assigning a grade of 7 is that scalability must be accompanied by a distributed file system. An example of such a file system is the HDFS. The HDFS enables *HBase* to achieve high scalability. However, the HDFS must be installed and set up. A problem with the scalability of

HBase is found when multiple clients update the same row or column at the same time, which leads to contention between the different clients. However, the reading clients will always see a consistent last update. Therefore, a grade of 7 is assigned as a result of the additional effort needed to enable high levels of scalability within the column-family databases.

7.2.6 Conceptual data structure

Several sources, such as online user-generated content and businesses that run all day, have led to the creation of large volumes of structured, unstructured, and semi-structured data (Dimiduk et al., 2013). *HBase* is a schema-less store that does not have a predefined structure (Dimiduk et al., 2013). Consequently, it does not support a full relational data model. The data model supports dynamic control over the data layout and format (George, 2011). Therefore, *HBase* can store and work with large volumes of semi- and unstructured data.

HBase is commonly employed in use cases that deal with large volumes of data (Hecht & Jablonski, 2011). This database can store any data type that can be converted into a byte of arrays (HBase, 2007). Stored data could consist of strings, images, numbers, and any other objects that can be converted (George, 2011). However, it is not geared to handle transactional data. If a use case employs heavily linked data, such as transactional data, *HBase* may not be the best solution to the problem. A popular use case for this type of database is storing Facebook messages (Aiyer et al., 2012). Facebook messages contain several types of data, including images, videos, and text. Other use cases include real-time analytics, monitoring systems, and search indexing (Aiyer et al., 2012). Additionally, *HBase* employs a storage technology known as HDFS to assist with the storage of the data.

The Hadoop Distributed File System (HDFS) provides highly scalable and reliable storage for data. Implementing the HDFS allows IT practitioners to control various aspects of the data, such as the data structure, so that semi- or unstructured data formats can be stored. Thus, *HBase* combined with HDFS allows the storage of semi- or unstructured data formats (George, 2011).

Ultimately, *HBase* can be used to store and work with large volumes of records coming from several sources (Dimiduk et al., 2013). The flexible schema of *HBase* allows the data to evolve over time. Therefore, a grade of 8 reflects the ability of column-family stores to work with large volumes of semi- or unstructured data.

7.2.7 Reliability

In a business environment, reliability is a key feature that can influence the operation of a business. Reliability in a database sense can refer to a system's ability to operate without failures for a certain amount of time (Domaschka, Hauser & Erb, 2014).

HBase employs the HDFS as storage mechanism and assumes that two properties of the HDFS will assist in providing reliability to the clients. The first property is *single namespace*, which refers to the single file system *HBase* uses to store data. It is assumed that all RegionServers across the entire cluster have access to the file system. The file system provides a single namespace, which the RegionServers must use to access the data. If data is visible or written by one RegionServer, all other RegionServers have access to that data. Therefore, *HBase* can make reliability guarantees to the entire cluster. If a RegionServer experiences a fault and goes down, the other RegionServers can access the data and serve the regions under the failed RegionServer (Dimiduk et al., 2013, p.79).

The second property is *reliability and failure resistance*, which refers to the assumption that the data in the underlying storage system will be accessible even after a failure occurs. If a RegionServer experiences a failure, the other RegionServers must be able to fulfil the role of the failed RegionServer. The assumption is that a failed RegionServer would not lead to downtime or data loss. A way downtime or data loss can be mitigated is through the HDFS replicating the data to other nodes and keeping copies of the data (Dimiduk et al., 2013, p.81). Therefore, column-family stores can guarantee certain levels of reliability.

The combination of column-family stores and the HDFS allows high reliability to be implemented. A grade of 7 is provided to *HBase* for reliability, since high reliability can be implemented within column-family stores. The underlying storage system of *HBase* is the HDFS. The HDFS allow an entire cluster to have access to the entire dataset in storage. A RegionServer should be able to access other RegionServers' data to serve their regions. It is assumed that the other RegionServers will serve a failed RegionServer’s regions. Therefore, a grade of 7 is assigned to column-family stores, as no downtime or data loss should occur. However, downtime or data loss is still possible.

7.2.8 Learning curve

The learning curve criterion refers to the time and effort needed for and complexity level of setting up and learning how to use a database that meets specific requirements. This criterion is hard to measure, since not all use cases are the same. Thus, the focus is placed on the volume of information that can be utilised. The information that will be investigated includes documentation data, books, and tutorials. These information sources can be used to teach IT practitioners about the column-family stores.

HBase in Action (Dimiduk et al., 2013) contains 329 pages that give the reader an overview of how this database works and what technologies it employs. *HBase in Action* (Dimiduk et al., 2013) starts by discussing the fundamentals of column-family stores and *HBase*. Thereafter, it describes how to install a single instance of *HBase*. *HBase in Action* teaches the reader how to start a fresh installation

of the database by providing tutorials and examples of code. The book introduces the reader to a variety of terms that refer to technologies and meeting the requirements of a use case. The book moves past a single instance setup of *HBase* to investigate *HBase* in a distributed environment and determine what the requirements of such an environment are. The book also begins to explain more advanced usage of *HBase*, such as table designs, and extensions of *HBase*. *HBase in Action* also has example applications to help teach the reader how to efficiently employ *HBase* to facilitate their use case.

The *HBase Administration Cookbook* (Jiang, 2012) teaches the reader how to set up *HBase* through tutorials and commands. This book has 315 pages that cover the most basic to the most advanced usage and setup of *HBase*. The *HBase Administration Cookbook* contains many code examples with explanations to teach the reader how to set up *HBase*. The book focusses on the administration of *HBase* clusters and how to tweak *HBase* to meet the requirements of the reader. This book caters to more experienced readers who want to learn about the advanced usage and setups of *HBase*.

The *HBase* website is where an individual can find all relevant documentation regarding *HBase*. A download link can be used to get the files necessary to install *HBase*. On the website (https://hbase.apache.org), up-to-date information regarding the latest release of *HBase* as well as all *HBase* documentation can be found. On YouTube (www.youtube.com), there are several tutorials for beginners and advanced users. Below is a list of popular books that can be used to teach users about column-family stores and *HBase*. These books can teach users the most basic terms and concepts as well as the most advanced usage of *HBase* and column-family stores.

- George, L. (2011). *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. Sebastopol, CA: O'Reilly Media, Incorporated.
- Jiang, Y. (2012). *HBase Administration Cookbook: master HBase configuration and administration for optimum database performance*. Birmingham, UK: Packt Publishing.
- Redmond, E., Wilson, J. R., & Carter, J. (2012). *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Dallas, TX: Pragmatic Bookshelf.
- Dimiduk, N., Khurana, A., Ryan, M. H., & Stack, M. (2013). *HBase in action*. Shelter Island, NY: Manning Publications Company.
- Shriparv, S. (2014). *Learning HBase: learn the fundamentals of HBase administration and development with the help of real-time scenarios*. Birmingham, UK: Packt Publishing Ltd.
- Garg, N. (2014). *HBase essentials: a practical guide to realizing the seamless potential of storing and managing high-volume, high-velocity data quickly and painlessly with HBase.* Birmingham, UK: Packt Publishing.
- Kerzner, M., & Maniyam, S. (2014). *HBase Design Patterns*. Packt Publishing Ltd.
- Vohra, D. (2016). *Apache HBase Primer*. Berkeley, CA: Apress.

Performing a search on the databases of Web of Science, IEEE Xplore, and ScienceDirect as well as on Google Scholar returned the following number of results. The keyword used for the search was "HBase".

- Web of Science: 55 results

- IEEE Xplore: 275 results
- ScienceDirect: 570 results
- Google Scholar: 24 200 results

A wide variety of materials can be used to teach IT practitioners about column-family stores. These materials include books with tutorials, video tutorials, and courses on *HBase*. The material is readily available, and *HBase*'s documentation is available on their website. The amount of documentation is always expanding as new information is added to the current body of knowledge. Considering the number of search results and the volume of learning materials available, column-family stores score a grade of 8 for the learning curve criteria. The rating of 8 means that individuals can download information, install the database, and teach themselves how to use it easily with the help of online tutorials and books. There are also tutorials and books that specifically show individuals how to set up *HBase* for and use advanced techniques. The results of the search in the databases show that research is being done to improve *HBase*, as most of the results were relevant research.

7.3 Document-based stores (*MongoDB*)

Document-based stores store key to value pairs in files known as documents. Within a document, the key for each value must be unique (Hecht & Jablonski, 2011). The documents in a database are grouped into collections (Abramova & Bernardino, 2013). Each document within the collection has a special ID key to identify that specific document. The ID key must be unique within the collection so that documents can be identified individually. Each value in a document is open for queries (Hecht & Jablonski, 2011). Complex data structures can be handled more conveniently because document-based stores allow several data types to be stored in a single document (Hecht & Jablonski, 2011). Document-based stores are developer-friendly, as they support multiple data types by being schema-free and do not place any restrictions on storing data (Hecht & Jablonski, 2011).

MongoDB provides a document querying mechanism that groups documents into collections (Abramova & Bernardino, 2013; Cattell, 2011). In *MongoDB*, the unique ID of each document in a collection can be specified. For instance, a unique ID can be the combination of a timestamp and the ID of a document. Figure 7.5 is a graphical representation of a document with values.

Figure 7.5: A representation of a document containing data (MongoDB, 2008).

7.3.1 Consistency

MongoDB provides consistency through replication of the stored data. Replication is a way of storing identical copies of the data on numerous servers to keep the data safe from faults (Banker, 2011; Chodorow, 2013). In this database, replication is set up through replica sets, which are groups of servers that consist of one primary and multiple secondaries (Banker, 2011; Chodorow, 2013; MongoDB, 2008). Figure 7.6 graphically depicts a replica set.

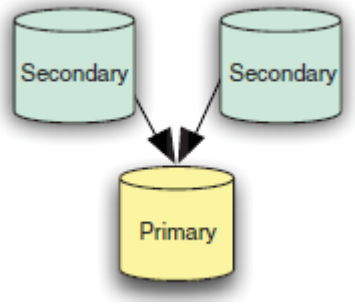

Figure 7.6: Graphical representation of a replica set (Banker, 2011).

The primary server accepts all the client requests, while the secondaries store copies of the primary's data. If a primary server experiences a fault, the secondary servers may elect a new primary server from the secondaries (Banker, 2011; Chodorow, 2013; MongoDB, 2008). Replication enables the client to access the full set of data even if the primary server fails, because full consistent copies of the data can be found on the secondaries. If the data on one server is damaged or corrupt, a new copy can be made from another server in the set (Banker, 2011, p. 10, 157; Chodorow, 2013, p. 169).

MongoDB can support strong consistency with multiple levels of consistency control (Chodorow, 2013; Hecht & Jablonski, 2011). The levels of consistency produce the configuration in which consistent data is displayed (Chodorow, 2013). A client might want to see only their own writes or request the most up-to-date data. To facilitate a high level of consistency, the server creates a que of requests for each connection (Chodorow, 2013), which represents the order in which the requests are performed. Any new requests will be placed at the back of the que (Chodorow, 2013), thereby enabling the connection to have a consistent view of the data.

There is a drawback to this method of consistency that caused by each connection having its own que of requests (Chodorow, 2013). If two simultaneous connections are made and one performs an insert and the other performs a read, the read operation may not be provided with the latest inserted document. Therefore, the client is presented with out-of-date data (Chodorow, 2013; Orend, 2010). Another problem that may occur is that when read requests are sent to a secondary server, the secondary may read old data (Chodorow, 2013). A solution to this problem is to forward all read requests to the primary server in the set.

Abramova and Bernardino (2013) indicate that *MongoDB* can provide good consistency using Master/Slave-like replication through the replica sets. According to their study (Brewer, 2012), *MongoDB* possesses the CP combination of the CAP theorem properties, which refers to consistency and partition tolerance. Therefore, strong consistency is achievable with this database.

Document-based stores score a grade of 7 for consistency. A grade of 7 means that high consistency can be achieved. However, there are some drawbacks to the method of consistency that *MongoDB* employs. The replica sets work in a manner comparable to Master/Slave replication. Therefore, downtime is possible when an election occurs. However, replica sets attempt to mitigate these drawbacks by immediately electing a new primary. Another potential problem with replica sets is that multiple concurrent connections can lead to inconsistent and out-of-date data being read.

7.3.2 Availability

Availability refers to the percentage of time a system is operating correctly (Orend, 2010). As described above, replica sets are used to provide strong consistency. However, replication and replica sets also influence availability.

In *MongoDB*, a replica set is set up to assist with fault tolerance (Banker, 2011; Chodorow, 2013). High availability is achieved using automatic failover within the replica sets (MongoDB, 2008). Automatic failover refers to a process during which an election occurs among the secondary servers (Banker, 2011; Chodorow, 2013). Automatic failover occurs when a primary server experiences a fault and goes down. The failover process means that a secondary server must be elected become the new primary (MongoDB, 2008). An improved version of the replication process is present from version 3.2 of *MongoDB*. The improved versions reduce the failover time and can detect if there are multiple primary servers in the set (MongoDB, 2008).

If replication is implemented, the client should be able to access the data even after a server goes down. All the servers in the set have access to the other servers' data. Thus, the client can still access the data even after a primary server goes down (Chodorow, 2013, p. 169).

Lourenço et al. (2015a) assigned a rating of "-" for availability to *MongoDB*, which means that this database may not be the best database to provide availability. A drawback of replica sets is that rollbacks can occur (MongoDB, 2008). A rollback reverts write operations on failed primary servers after a failover process occurs (MongoDB, 2008). Rollbacks occur when failed primary servers that have come online again re-join the replica set. However, rollbacks do not always occur after a failover. Rollbacks only occur when the primary server contained write requests that the secondary servers had not yet replicated when the primary stepped down (MongoDB, 2008). Rollbacks occur to maintain consistency between the new secondary server and the other servers (MongoDB, 2008).

Each rollback cannot roll back more than 300 megabytes of data. Therefore, manual recovery is required for rollbacks of more than 300 megabytes (MongoDB, 2008).

MongoDB provides an adequate availability process with automatic failover. The process has limits and resembles the Master/Slave replication method but has fewer drawbacks. However, data can be lost if rollbacks occur on more than 300 megabytes of data. This will require manual recovery to retrieve the data. The failover process can provide good levels of availability, but it is not the best option. Therefore, document-based stores receive a grade of 5 for availability.

7.3.3 Partitioning

Abramova and Bernardino (2013) compared the performance of *MongoDB* and *Cassandra*. In their study, it is shown that *MongoDB* is of the CP type, which means that partitioning is a focus within this database.

There are two strategies that can be followed when implementing partitioning (section 5.3.4). The first strategy is to distribute datasets by the range of their keys. This is known as range-based partitioning (Hecht & Jablonski, 2011). The second strategy is consistent hashing (Hecht & Jablonski, 2011).

The documents in *MongoDB* are partitioned by the range of their keys (range-based partitioning) (Banker, 2011; Hecht & Jablonski, 2011). Hecht and Jablonski (2011) investigated the partitioning performance of NoSQL databases as part of their study. *MongoDB* received a positive rating (+) for range-based partitioning and a negative rating (-) for consistent hashing. These ratings mean that it performs range-based queries well. This is because neighbouring keys are stored next to one another.

High partition tolerance in this database is achieved through the range-based partitioning strategy known as sharding (Banker, 2011). Sharding splits data across numerous servers. This is done by storing subsets of data on other servers in a cluster (Chodorow, 2013). Sharding can be implemented in two ways, namely manual sharding and autosharding (Banker, 2011; Chodorow, 2013). Manual sharding occurs when an application connects to several independent databases. The client application manages the storing of data on different servers as well as the queries to retrieve data. The manual sharding approach can work well. However this approach can struggle with the addition or removal of nodes within a cluster (Banker, 2011; Chodorow, 2013, p. 231).

Autosharding attempts to automate sharding by simplifying the administration process (Chodorow, 2013). *MongoDB* employs autosharding and allows the client application to communicate with the whole cluster as opposed to one server. Autosharding allows the addition or removal of nodes, while balancing the data across the servers (Banker, 2011; Chodorow, 2013, p. 231)

MongoDB uses autosharding, which is the superior option for providing partition tolerance. Setting up autosharding is a troublesome process, and extensive knowledge is required to set up all components correctly. Setting up *MongoDB* with autosharding is difficult and complex. However, setting up this database with autosharding allows better communication with the whole cluster by balancing the data across the entire system. Autosharding makes the addition and removal of nodes easier. Therefore, achieving high partition tolerance is also made easier.

As a result of the above-mentioned points, *MongoDB* receives a grade of 7 for partitioning and partition tolerance. *MongoDB* employs the range-based partitioning strategy to store neighbouring documents on the same node. This leads to better performance with queries. Autosharding allows easy addition and removal of nodes. However, extensive knowledge is required to set it up properly.

7.3.4 Read and write performance

Du Toit (2016) performed bulk inserts of various amounts of records (ranging from 100 to 20 000) into *MongoDB*. Du Toit's (2016) research indicated that the time *MongoDB* took to insert a record remained constant even if the dataset size increased. Figure 7.7 shows the performance results of *MongoDB*'s bulk insert test (Du Toit, 2016). Du Toit (2016) noted that *MongoDB* took an average of 2.5 to 3.5 milliseconds to insert a single record. The addition of other nodes did not decrease the insertion time (Du Toit, 2016). At 20 000 records, there was a spike in the insertion time for one of the nodes. According to the author, this increase occurred because *MongoDB* optimised itself based on the client application and what the application was doing (Du Toit, 2016).

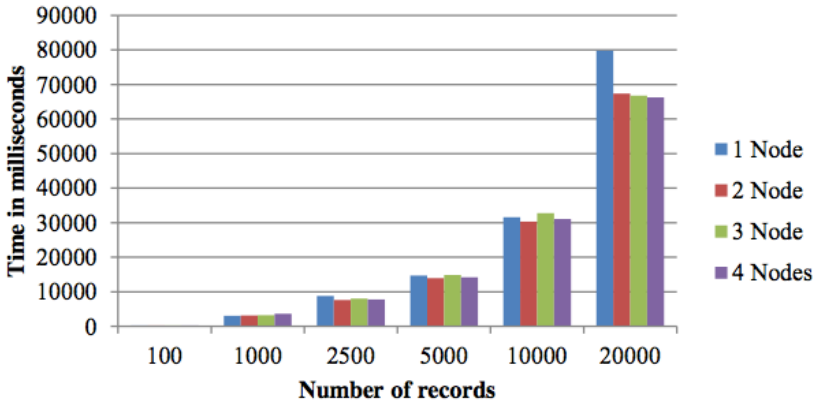

Figure 7.7: *MongoDB* data insert averages over four nodes (Du Toit, 2016).

Du Toit (2016) performed data read tests on *MongoDB* with various amounts of records ranging from 1 to 50 000 records. The data write tests could only accept up to 20 000 records compared to the 50 000 that were accepted by the read tests (Du Toit, 2016). This indicates that *MongoDB* can perform read operations better than write operations. Completing the 100-record job took 2.1 milliseconds, whereas it took 2.35 milliseconds for 10 000 records (Du Toit, 2016). *MongoDB* allows

result sets of 20 000 records to be returned by default (Du Toit, 2016). A result set of 20 000 records took an average of 3.8 seconds per record to complete. His research also shows that *MongoDB* reads a document set of 50 000 records in 179 seconds, compared to the 251 seconds it takes to insert them. Therefore, his research indicates that *MongoDB* is optimised for reading purposes (Du Toit, 2016).

During the investigation of *MongoDB*'s read performance in Du Toit's (2016) research, the retrieval of a single record took 17.6 milliseconds on average. His research indicates that the average retrieval time for single records was higher than the average retrieval time for records that were part of larger read requests (Du Toit, 2016). His research also shows that the size of the result set impacted the performance (Du Toit, 2016). According to his research, the best performance was recorded when retrieving a 1000 records. A speed of 2.414 milliseconds per record was achieved (Table 7.7) (Du Toit, 2016). *MongoDB* was able to return the larger result set without using batches.

Table 7.7: *MongoDB* read statistics (Du Toit, 2016).

Records	1	100	1 000	2 500	5 000	10 000	20 000	50 000
Duration in ms (average)	17.6	314.8	2 414	6 089.2	12 610.9	25 378	56 114.8	176 203
Latency per record in ms	17.6	3.148	2.414	2.43568	2.52218	2.5378	2.80574	3.52406
Records per second	56.8182	317.662	414.25	410.563	396.482	394.042	356.412	283.764

Lourenço et al. (2015a) made a comparison of NoSQL databases using a set of properties. The authors assigned ratings to these properties to indicate the performance level of each database. In their rating system, *MongoDB* received a rating of "-" for write performance and "++" for reading performance (Lourenço et al., 2015a). A rating of "++" indicates that *MongoDB* is very focused on reading performance. This implies that *MongoDB* is read optimised.

Győrödi et al. (2015a) compared the performance of *MongoDB* and *MSSQL*. In their study, they proved that there are major differences between these two types of databases (NoSQL vs. SQL). The authors indicated that *MongoDB* took less time to read 50 000 records than to write 50 000 records. Figure 7.8 is a graphical representation of the results for the read and write tests.

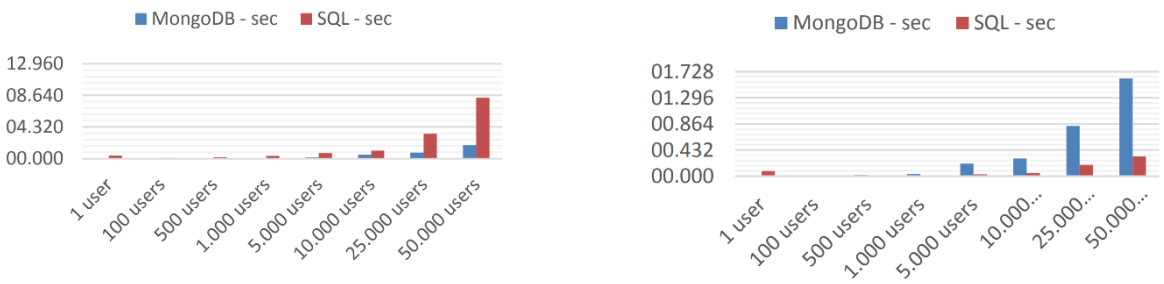

Figure 7.8: Read and write performance of *MongoDB* (Győrödi et al., 2015a).

Győrödi et al. (2015b) also compared *MongoDB* with *MySQL*. Experiment tests were done to compare these two databases with each other (Győrödi et al., 2015b). The first test was to write to the databases. The authors inserted 10 000 records into *MongoDB* (Győrödi et al., 2015b). *MongoDB* took 0.29 seconds to complete the write operation (Győrödi et al., 2015b). The second test was to read from the database. *MongoDB* took 0.0052 seconds to complete the read operation (Győrödi et al., 2015b). The reading test results show that this database has fast reading performance. Comparing the writing time (0.29 seconds) with the reading time (0.0052 seconds) indicates a significant gap in performance levels. Therefore, the results imply that *MongoDB* is read optimised. Figure 7.9 shows a graphical representation of the results for its reading and writing performance tests.

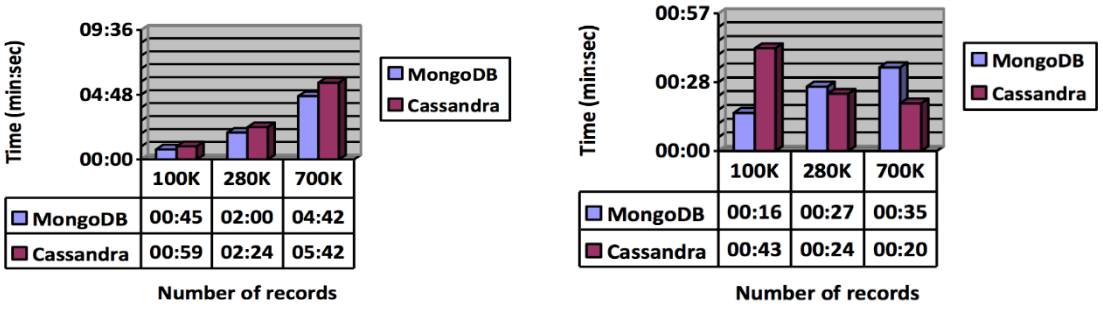

Figure 7.9: Graphical representation of the performance test results (Győrödi et al., 2015b).

Abramova and Bernardino (2013) compared *MongoDB* and *Cassandra*. Their study shows that *MongoDB* can provide good reading and writing performance. The amounts of records used in their tests were 100 000, 280 000, and 700 000 (Abramova & Bernardino, 2013). A comparison of the read and write results indicates the superiority of read performance over write performance as was the case throughout all the performance tests. At the 700 000-record test, *MongoDB* completed the read operation in 35 seconds, while the write operation took 282 seconds. These results support the previously mentioned studies that indicated the read optimisation of *MongoDB*. Figure 7.10 indicates the results of the write and read performance tests.

Figure 7.10: Write and read speeds of *MongoDB* compared with *Cassandra* (Abramova & Bernardino, 2013).

Li and Manoharan (2013) proved that *MongoDB* is exceptional with reads and average with writes. The authors tested the read and write performance of *MongoDB* against several other NoSQL products. *MongoDB* was one of the best databases in each of their experiment. Figure 7.11 compares the reading performance of the several database products. The time took to complete the operations was measured in milliseconds. The number of operations refers to the number of times a specific operation is executed and ranged from 10 to 100 000 (Li & Manoharan, 2013). The results of the performance tests show that *MongoDB* had the second fastest read performance of the databases that were tested.

Database	Number of operations					
	10	50	100	1000	10000	100000
MongoDB	8	14	23	138	1085	10201
RavenDB	140	351	539	4730	47459	426505
CouchDB	23	101	196	1819	19508	176098
Cassandra	115	230	354	2385	19758	228096
Hypertable	60	83	103	420	3427	63036
Couchbase	15	22	23	86	811	7244
MS SQL Express	13	23	46	277	1968	17214

Figure 7.11: Reading performance of databases (Li & Manoharan, 2013).

Figure 7.12 shows the write performance of the several databases. A comparison of the reading and writing performance results for *MongoDB* indicates a difference in performance levels. At 100 000 operations, reading took 10201 milliseconds, while writing took 23354 milliseconds. The writing performance test took more than double the time of the reading performance test. Therefore, the results indicate that reading performance is far superior to writing performance and that *MongoDB* is read optimised.

Database	Number of operations					
	10	50	100	1000	10000	100000
MongoDB	61	75	84	387	2693	23354
RavenDB	570	898	1213	6939	71343	740450
CouchDB	90	374	616	6211	67216	932038
Cassandra	117	160	212	1200	9801	88197
Hypertable	55	90	184	1035	10938	114872
Couchbase	60	76	63	142	936	8492
MS SQL Express	30	94	129	1790	15588	216479

Figure 7.12: Writing performance of databases (Li & Manoharan, 2013).

Considering the results of all the above-mentioned studies, it may be concluded *MongoDB* is read optimised (Abramova & Bernardino, 2013; Du Toit, 2016; Győrödi et al., 2015a; Győrödi et al., 2015b; Li & Manoharan, 2013; Lourenço et al., 2015a). Read optimisation means that the reading performance is better than the writing performance. Thus, a grade of 9 is assigned for reading performance and a grade of 5 for writing performance. These ratings reflect the gap between reading and writing performance within *MongoDB*. *MongoDB* can accommodate a high read request use case

with a moderate number of writes. If the use case is write heavy, then *MongoDB* might not be a suitable choice for optimal performance.

7.3.5 Scalability

Partitioning data across multiple servers is the way document-based databases provide scalability (Cattell, 2011). *MongoDB* allows scaling of data across multiple servers in a distributed environment by employing automatic sharding (Banker, 2011; Chodorow, 2013; MongoDB, 2008). Replication in *MongoDB* is used for redundancy purposes and not for scalability reasons. However, a Master/Slave-like replication model is used (Du Toit, 2016).

In *MongoDB*, a mongod instance allows data to shard across several database nodes in a cluster (MongoDB, 2008). Mongod stands for mongo daemon. The mongod instance is responsible for storing the subset of the collection's data (Figure 7.13) (Chodorow, 2013; MongoDB, 2008). Config servers are also a requirement for storing the metadata of the clusters (MongoDB, 2008). The config server can be found in a mongod instance (MongoDB, 2008). The queries from clients are directed to the appropriate shard on a mongod instance via the mongos routing service (Chodorow, 2013; MongoDB, 2008). *MongoDB* instances on each node start as soon as the config and routing servers are running. Thereafter, they are added to the cluster through the routing service (MongoDB, 2008).

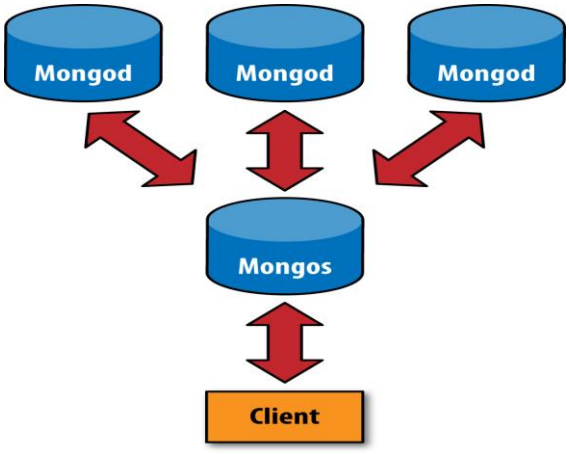

Figure 7.13: Graphical representation of a sharded client connection (Chodorow, 2013, p.233).

Du Toit's (2016) research found that data in *MongoDB* is not sharded automatically. To shard data, an index key must be specified (MongoDB, 2008). He also found that performance increases can be noticed when adding nodes to the cluster (Du Toit, 2016). If autosharding is set up within *MongoDB*, it will manage the distribution of data across the nodes as well as facilitate the addition of nodes to the cluster (Banker, 2011). The document data model allows documents in *MongoDB* to be divided between the different nodes in the cluster (Chodorow, 2013). In the rating system of Lourenço et al. (2015a), *MongoDB* received a rating of "-" for scalability, implying that *MongoDB* might not the best choice to facilitate high scalability.

As a result of the aforementioned points, a grade of 6 is assigned to document-based stores for scalability. A grade of 6 means that document-based stores may not provide the best levels of scalability. However, document-based stores can provide adequate levels of scalability across the cluster. *MongoDB* employs sharding to facilitate its scalability, which is difficult to set up in the correct manner. There are plenty of components to be set up that require extensive knowledge pertaining to the functioning of sharding. Sharding is not done automatically when *MongoDB* is set up. However, good scalability can be achieved through autosharding. Autosharding makes the scalability process much easier if it is set up correctly to handle the distribution of data across the nodes automatically. Autosharding also manages the queries sent to the nodes. Therefore, clients are not aware that they are communicating with other nodes. The above reasons are why a grade of 6 is assigned to document-based stores.

7.3.6 Conceptual data structure

Document-based stores can store semi- and unstructured data (Banker, 2011; Chodorow, 2011). This family employs a flexible data model with no predefined schema (Banker, 2011; Chodorow, 2013; Orend, 2010). A document-based store, such as *MongoDB*, groups documents into collections (Banker, 2011). A document refers to the basic unit of data for *MongoDB*. A document contains a key and the value(s) associated with that key. Collections store documents with a similar data structure (Banker, 2011). Multiple data structures can be stored within a single collection. A single instance of *MongoDB* can have multiple separate databases, each with their own collections inside. (Chodorow, 2013, p. 7).

There are advantages to employing a schemaless data model. Firstly, it is easier to make changes to the dataset (Banker, 2011). Secondly, it is easier to add or remove fields (Chodorow, 2013). Lastly, the schemaless model allows the representation of several data types within a single document (Banker, 2011). Use cases for document-based stores are typically involve storing enormous-size collections of documents (Moniruzzaman & Hossain, 2013). Use cases include high volume data feeds, operational intelligence, behavioural profiling, content management, and metadata storage (Magnusson, 2013). Some data types stored within documents include text, emails, XML, and semi-structured data (Moniruzzaman & Hossain, 2013). Document-based stores are geared to work with semi-structured data (Kaur & Rani, 2013). However, they are not geared to deal with relationship-heavy data.

Thus, document-based stores receive a grade of 7 for the conceptual data structure criterion. A grade of 7 represents the ability of document-based stores to work with semi- and unstructured data. The database provides a schemaless storage model for the data, which means that various data structures can be stored together. The schemaless model enables changes to the data to be made easily without

affecting the database. The use cases indicate some of the different data types, each with their own structure, that can be stored within this family. However, it is not well suited for relationship-heavy data, such as transactional data.

7.3.7 Reliability

As stated previously, the reliability of a system refers to its ability to operate without faults that reduce the operation quality (refer to section 5.3.7). Within *MongoDB*, replication can provide reliability to the system through replica sets (Banker, 2011; Plugge, Hows, Membrey, & Hawkins, 2015). A replica set consist of a primary node and two or more secondary nodes (Banker, 2011; Plugge et al., 2015). Replica sets combat faults through fault tolerance (Banker, 2011; Chodorow, 2013) and employ automatic failover to mitigate the effects of downtime and data loss. Automatic failover can also provide fault tolerance and high reliability (MongoDB, 2008).

If a primary server goes down, an election takes place to select a new primary server. Once the election process is completed, the elected secondary immediately becomes the new primary server and handles all the requests from clients. If replication is implemented and a primary server goes down, the data should still be accessible. All the servers in the set have access to the other servers' data. Therefore, the client has access to the data even if a primary server experiences a fault. If the data on one server is corrupt, a new copy can be made from the other servers in the set. (Chodorow, 2013, p. 169; Plugge et al., 2015). Thus, replication in *MongoDB* increases the reliability of the overall database deployment (Banker, 2011).

Another feature that can aid in increasing the reliability of the overall database system is RAID setups. RAID is software that allows one to handle multiple disks as if they were a single disk (Chodorow, 2013). A RAID array is a set of disks that implement RAID software. There are numerous levels of RAID and each level has distinctive features. The levels are RAID0, RAID1, RAID5, and RAID10. RAID10 is the best option for reliability, because the data is striped and mirrored. The level chosen for a specific use case depends on how reliable the database needs to be for that use case (Chodorow, 2013, pp. 369-370).

A grade of 7 is assigned for this criterion because document-based stores can provide high reliability. However, there are some drawbacks to the methods *MongoDB* employs to provide reliability. The replica sets allow reliability to be high, because the primary server is replaced as soon as it goes down. Replica sets improve this process by instantly electing a new primary server. However, data loss can still occur due to rollbacks. Another drawback is that these features (RAID and replica sets) need to be set up before reliability can be achieved. *MongoDB* does not have these features set up

automatically. A grade of 7 indicates that high levels of reliability can be achieved with document-based stores. However, this requires some effort.

7.3.8 Learning curve

The learning curve criterion refers to the time and effort needed to set up and learn how to use a database that meets specific requirements. This criterion is hard to measure, as not all use cases are the same. Therefore, the focus of this criterion is on the available knowledge regarding document-based stores. The investigated information includes books, tutorials, and official documentation. These sources of information can provide IT practitioners with all the relevant knowledge needed to employ document-based stores.

MongoDB in Action (Banker, 2011) contains 287 pages that aim to provide the reader with a holistic view of document-based stores and *MongoDB*. The book starts by explaining basic terms and concepts related to document-based stores and *MongoDB*. The book examines the techniques that *MongoDB* implements, for example the sharding technique. The book also assists the reader in setting up *MongoDB* in a basic single node environment through code examples. *MongoDB in Action* shows the reader how to manage and troubleshoot the database if faults occur. This book focusses on beginners who want to get started with *MongoDB*.

MongoDB: The Definitive Guide is a book written by Chodorow (2013) that helps readers with basic and advanced usage of *MongoDB*. This book has 409 pages that cover many aspects of the *MongoDB* database system. The book informs the reader about many relevant terms and technologies that *MongoDB* employs. Detailed instructions on how to set up and use *MongoDB* in a basic environment are provided. Advanced techniques, such as sharding and replication, are discussed in detail. It also explains how *MongoDB* employs these techniques to achieve a goal. Basic setups of the advanced techniques are described to the reader. Administration of the *MongoDB* server is covered in a large section of this book, which helps the reader with several relevant administration tasks. This book is appropriate for beginners as well as advanced users of *MongoDB*.

A book focused on advanced usage of *MongoDB* is the *MongoDB Cookbook* written by Nayak (2014). This book starts by explaining how to set up *MongoDB* through the use of code examples. The book covers single node setups as well as multi-node distributed setups. The book also covers advanced management of the database and dataset. Furthermore, it assists the reader in implementing and deploying *MongoDB* with Hadoop and other open source tools to accomplish tasks. This book is focusses on advanced usage of *MongoDB* and how to manipulate the database to accomplish tasks for a specific use case.

The *MongoDB* website (https://www.mongodb.com) is where a wide variety of resources, including all documentation concerning *MongoDB,* can be obtained. Research papers on *MongoDB* can also be retrieved from the website. A download link on the website allows clients to download and install *MongoDB*. There are also numerous tutorials for beginners and advanced users on YouTube (www.youtube.com). Below is a list of popular books about document-based stores and *MongoDB*. These books can teach users about *MongoDB* and how it functions. The books also explain how to set up a basic *MongoDB* instance and discuss advanced usage of *MongoDB*. The books can train users to become well informed about the commands used in *MongoDB*.

- Chodorow, K. (2013). *MongoDB: the definitive guide*. Sebastopol, CA: O'Reilly Media, Incorporated.
- Banker, K. (2011). *MongoDB in action*. Shelter Island, NY. Manning Publications Company.
- Plugge, E., Hows, D., Membrey, P., & Hawkins, T. (2015). *The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB*. California: Apress.
- Chodorow, K. (2011). *Scaling MongoDB*. Sebastopol, CA: O'Reilly Media, Incorporated.
- Copeland, R. (2013). *MongoDB Applied Design Patterns*. Sebastopol, CA: O'Reilly Media, Incorporated.
- Francia, S. (2012). *MongoDB and PHP: Document-Oriented Data for Web Developers*. Sebastopol, CA: O'Reilly Media, Incorporated.
- Chodorow, K. (2011). *50 tips and tricks for MongoDB developers*. Sebastopol, CA: O'Reilly Media, Incorporated.
- Marchioni, F. (2015). *MongoDB for Java Developers: design, build, and deliver efficient Java applications using the most advanced NoSQL database*. Birmingham, UK: Packt Publishing.
- Nayak, A. (2014). *MongoDB cookbook: over 80 practical recipes to design, deploy, and administer MongoDB*. Birmingham, UK: Packt Publishing.
- Hows, D., Membrey, P., & Plugge, E. (2014). *MongoDB basics*. Berkeley, CA: Apress.
- Nayak, A. (2013). *Instant mongodb*. Birmingham, UK: Packt Publishing.
- Edward, S. G., & Sabharwal, N. (2015). *Practical MongoDB: architecting, developing, and administering MongoDB*. New Delhi: Apress.
- Hoberman, S. (2014). *Data modeling for MongoDB: building well-designed and supportable MongoDB databases*. Basking Ridge, NJ: Technics Pub.
- França, W. D. R. (2015). *MongoDB data modeling*. Birmingham, UK: Packt Publishing.
- Mehrabani, A. (2014). *MongoDB high availability design and implement a highly available server using the latest features of MongoDB*. Birmingham, UK: Packt Publishing.
- Holmes, S. (2015). *Getting MEAN: with Mongo, Express, Angular, and Node*. Shelter Island, NY: Manning.
- Vohra, D. (2015). *Pro MongoDB development*. Apress.

Doing a search on the databases of Web of Science, IEEE Xplore, and ScienceDirect as well as on Google Scholar returned the following number of results. The keyword used for the search was "MongoDB".

- Web of Science: 44 results
- IEEE Xplore: 164 results

- ScienceDirect: 353 results
- Google Scholar: 18 100 results

Document-based stores have a variety of teaching materials readily available on the internet. Out of all the families, document-based stores have the most teaching documentation available. *MongoDB* has an organisational website (https://www.mongodb.com) on which the documentation for the database can be found. A download link is available to download the necessary program to install *MongoDB*. There are also book and video tutorials on how to set up and use *MongoDB* in different situations. The books about *MongoDB* can teach an individual the fundamentals of document-based stores as well as *MongoDB*. The teaching materials cater for beginners and advanced users of *MongoDB*. Document-based stores are assigned a grade of 9 due to the amount of available teaching materials for *MongoDB*. A grade of 9 means that individuals can easily teach themselves how to set up, use, and accommodate their use case through *MongoDB*. The database search results also indicate that research is being done to improve document-based stores and *MongoDB*.

7.4 Graph stores (*Neo4j*)

A graph, in formal terms, is a group of vertices, properties, and edges (Kemper, 2015). A graph can also be seen as a set of nodes and the relationships that connect them. When a node is created, it receives properties as well as any edges that are used (Kemper, 2015). Nodes represent the different entities in the graph, while relationships show their relation to one another. The structure of a graph allows different scenarios to be modelled. A graphical example of a graph is seen in Figure 7.14 (Robinson, Webber & Eifrem, 2015, p. 1).

Figure 7.14 represents a small group of individuals and the relationships they have with one another. It represents a small network of Twitter users and not the entire Twitter network. Each node represents a specific user. Users are connected to one another through their relationships (Robinson,

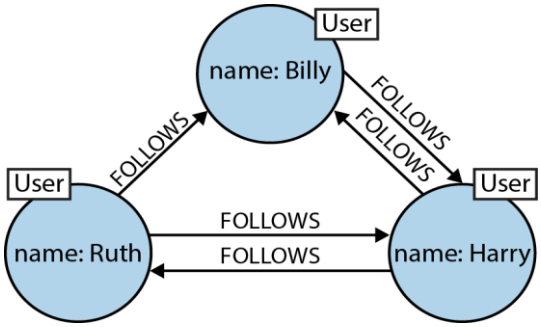

Figure 7.14: Graph example within the Twitter context (Robinson, Webber & Eifrem, 2015).

Webber & Eifrem, 2015, p. 2). The graph provides a holistic view of the stored data through a graphical representation that makes it easier to understand. *Neo4j* is graph database that allows developers to provide good performance where queries over large and complex datasets are concerned (Goel, 2015).

7.4.1 Consistency

Completing a write operation will insert a record into a database. If the database has high consistency, all readers will view the most up-to-date information (Brewer, 2012; Strauch, Sites & Kriha, 2011). Thus, consistency refers to the extent to which the system is in a consistent state after operations occur. *Neo4j* provides eventual consistency (Hecht & Jablonski, 2011), meaning that if no new updates to a record are made, all reads show the latest updated record. *Neo4j* employs Master/Slave replication to facilitate consistency throughout its clusters.

The typical Master/Slave replication setups require all write requests to operate through the Master, while read requests go to the Slaves (Vukotic et al., 2015). However, *Neo4j* does not employ the typical Master/Slave replication model. Therefore, any Master/Slave node can handle both reads and writes (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015). Writing to a Slave does have some drawbacks. To ensure its data is consistent, the Slave must synchronise with the Master through a coordination protocol before it can return to the client (Robinson, Webber & Eifrem, 2015). This process creates extra network traffic and causes the Slave nodes to be slower than the Master nodes. Master and Slave nodes handle write requests differently (Vukotic et al., 2015). Reasons to write to a Slave include that it can provide durability guarantees and enable clients to read their own writes (Robinson, Webber & Eifrem, 2015). It is recommended that writes be made only on the Master and then replicated to the Slaves (Robinson, Webber & Eifrem, 2015).

Updates to records are applied to the Master node first. If this is successful, then the updates are applied to the Slaves. A Slave must be up to date to ensure overall consistency of data between the Master and Slave nodes. If the Slave is up to date, then write requests can be performed. This means that *Neo4j* will make sure that all Slaves are up to date before local writes can occur (Vukotic et al., 2015, p. 238). This process can be seen as eventual consistency.

A grade of 6 is assigned to *Neo4j* for its ability to provide eventual consistency. A grade of 6 means that above-average consistency performance can be achieved. However, the consistency is not without faults. *Neo4j* provides full consistency in single instances but eventual consistency in a distributed environment. The Master/Slave replication method has its own drawbacks, such as downtime. Master/Slave replication assists in providing consistency of data. A Master node will always be present to handle the read and write requests. *Neo4j* also allows reading and writing to both Master and Slave nodes. However, writing to a Slave node leads to slower performance. The data on the nodes is eventually consistent, which means that clients will eventually read the most up-to-date information.

7.4.2 Availability

Availability refers to a system's resistance to faults that may occur and its ability to provide continuous operation (Han, Haihong, Le & Du, 2011; Orend, 2010; Strauch, Sites & Kriha, 2011). An example of a fault occurring is a node in a cluster going down. Availability in *Neo4j* is achieved through a component known as *Neo4j* HA (high availability). The HA component allows the database to run in a clustered setup. This enables the distribution of the database across multiple machines. A Master/Slave replication architecture is employed to provide fault tolerance and resistance to failures (Montag, 2013; Vukotic et al., 2015). Figure 7.15 is a graphical depiction of the Master/Slave replication used to provide fault tolerance.

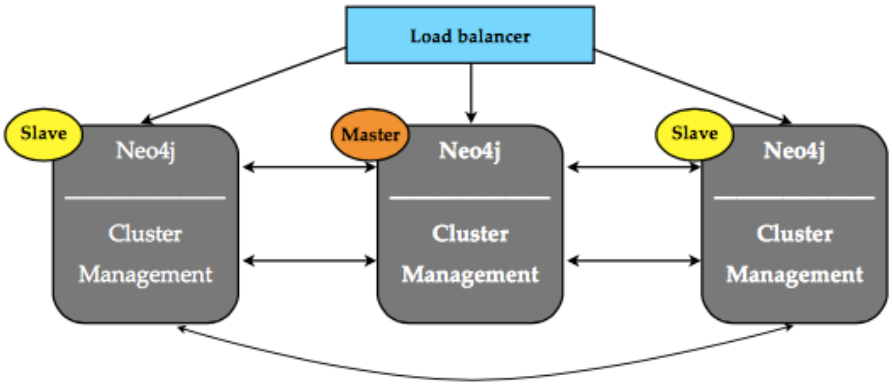

Figure 7.15: A graphical representation of the Master/Slave replication architecture of *Neo4j* (Montag, 2013).

Using a Master/Slave replication architecture allows *Neo4j* to counter hardware failures as well as handle large volumes of read requests (Vukotic et al., 2015). Each *Neo4j* instance contains two parts, namely the database and the cluster management component (Montag, 2013). The cluster management component is synchronised with all the instances in the database to keep track of instances that join and leave the cluster (Montag, 2013). Therefore, if the database experiences a fault, such as a hardware failure or a network outage, the cluster management component will detect the failure and mark the database as having temporarily failed (Montag, 2013; Neo4j, 2017). The database will update itself with the rest of the cluster when it comes back online (Neo4j, 2017).

The cluster management component works with the Master/Slave replication method. Within the cluster, it is expected that a single Master will always be present along with any number of Slaves (Vukotic et al., 2015). If a Master goes down, the cluster management component will ensure that a Slave is elected as the new Master (Montag, 2013; Neo4j, 2017). The new Master begins to function after a quorum is reached. This means that more than half of the cluster members must be active (Neo4j, 2017). The new Master will broadcast its availability to all the members of the cluster (Neo4j, 2017). Typically, the election of a new Master will occur within a few seconds. However, no new writes can be accepted during the election time (Neo4j, 2017).

An advantage of Master/Slave replication is the ability to write through both Master and Slave nodes (Montag, 2013). Although the write performance of Slave nodes is not the best, clients can still write to the database. Therefore, *Neo4j* can provide availability to the clients.

A grade of 9 is assigned to *Neo4j* for its availability property. The Master/Slave replication architecture enables very high uptime. However, there are drawbacks to employing the Master/Slave replication architecture. The election of a new Master can lead to downtime or data loss, since the election process prohibits clients from writing to the database. The load balancer assists with the election process and combats the drawbacks to a certain degree. The load balancer enables *Neo4j* to have an automatic election whenever a Master node goes down. Therefore, *Neo4j* is geared to provide high availability to the cluster. The HA component supports resilience and fault tolerance to ensure the availability of the database.

7.4.3 Partitioning

Partition tolerance refers to the ability of a system to continue to function even if there are faulty network partitions (Strauch et al., 2011). In data partitioning situations, partition tolerance plays a vital role in ensuring continuous operation.

Queries are executed the quickest when the graph dataset is stored in main memory (Robinson, Webber & Eifrem, 2015). However, the size of main memory becomes a problem when the graph dataset is too large to store in main memory. Partitioning plays a vital role, as other technologies partition their data to solve this problem (Robinson, Webber & Eifrem, 2015).

A graph dataset is difficult to divide among several partitions, because partitions may influence the data and the relationships between the entities (Hecht & Jablonski, 2011). Thus, a problem is faced when partitioning a graph dataset. On the hand, partitions provide better performance and fault tolerance. On the other hand, a heavily linked graph dataset should not be distributed, because traversals and lookups would cause performance penalties (Hecht & Jablonski, 2011) due to the additional heavy network load.

As mentioned above, partitioning a graph dataset is not an easy task. Cache sharding can solve the problem to a certain degree (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015). In a *Neo4j* cluster with the HA component, each HA instance expects to have access to the full set of data. Cache sharding is a routing-based technique that routes requests to a certain database instance within the *Neo4j* HA cluster (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015). Cache sharding routes requests to the specific database instance that can best satisfy them (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015). Therefore, it assists in increasing the performance of requests over a distributed environment.

Partitioning a graph dataset is a challenging task to accomplish, because the graph can rapidly mutate and grow in size and relationships. The node relationships play a significant role in creating the value that a graph database provides. Partitioning a large graph dataset across multiple nodes can incur performance penalties and decrease the value of the dataset. Thus, partition tolerance is assigned a grade of 4 for graph stores. Partitioning a large graph dataset is not recommended, because a large number of relationships can lead to complex queries. However, cache sharding is a technique that can be used to assist in partitioning data in graph databases.

7.4.4 Read and write performance

Jouili and Vansteenberghe (2013) developed a benchmark that can be used to test the performance levels of graph databases by running simulations of real graph workloads. Their tool was used to test *Neo4j*'s write performance. Jouili and Vansteenberghe (2013) also investigated the effect a bigger buffer size has on the writing performance of graph databases. The buffer size refers to the number of records inserted before the records are stored on a disk (Jouili & Vansteenberghe, 2013). They found that the larger the buffer size, the better the performance of *Neo4j*.

The writing performance results (Figure 7.16) indicate that *Neo4j* achieved the best write performance results in the performance tests until the 2.5 million-record entry. Thereafter, *Neo4j*'s time increased drastically from 33.10 seconds to 297.87 seconds (Jouili & Vansteenberghe, 2013), which was the second slowest time.

Graph el. loaded	Neo4j	DEX	Titan-BDB	Titan-Cassandra	OrientDB
500k	5.74995	19.44911	18.53484	52.51524	56.75998
1000k	12.04847	32.03252	33.23480	81.62522	588.83879
1500k	16.98259	51.85323	50.31449	115.72004	1005.19257
2000k	22.15680	66.73069	65.55982	139.48701	1359.54699
2500k	33.10645	82.25261	80.15614	178.41247	1986.79631
3000k	297.87892	99.1784	94.44010	215.78923	4350.20323

Figure 7.16: Workload results using a buffer size of 5000 records (Jouili & Vansteenberghe, 2013).

Next, the authors investigated the effect of increasing the buffer size on the write performance (Figure 7.17). The buffer size increased from 5000 to 20 000. *Neo4j* completed the 3 million-record workload in a time of 143.57 seconds, compared to its previous time of 297.87 seconds (Jouili & Vansteenberghe, 2013). The performance results may imply that *Neo4j* provides good linear write performance up to a certain point. However, after that point is reached, the performance decreases and is not linear anymore. 17Workload results using a buffer size of 20 000 (Jouili & Vansteenberghe, 2013). 18Workload results using a buffer size of 20 000 (Jouili & Vansteenberghe, 2013).

Graph el. loaded	Neo4j	DEX	Titan-BDB	Titan-Cassandra	OrientDB
500k	3.98751	19.06939	19.52635	50.56084	54.59390
1000k	8.25654	30.99353	31.43026	80.49548	588.90165
1500k	11.06476	44.76818	44.60898	118.12311	1010.05765
2000k	14.02776	64.67971	55.74031	144.46303	1388.35480
2500k	18.81234	79.394	66.35664	171.19640	1808.26900
3000k	143.57329	95.60929	77.19206	206.45922	5552.70849

Figure 7.17: Workload results using a buffer size of 20 000 (Jouili & Vansteenberghe, 2013).

The read test results of *Neo4j* indicate good reading performance. It was the top database in the set of databases. As shown with the Figure 7.18, *Neo4j* had the best results while also having the least variance in results (Jouili & Vansteenberghe, 2013). The results of this study found that *Neo4j* is a top contender with good performance for both reading and writing jobs. Based on these results, it can be concluded that the reading performance of *Neo4j* is better than its writing performance. This indicates read optimisation.

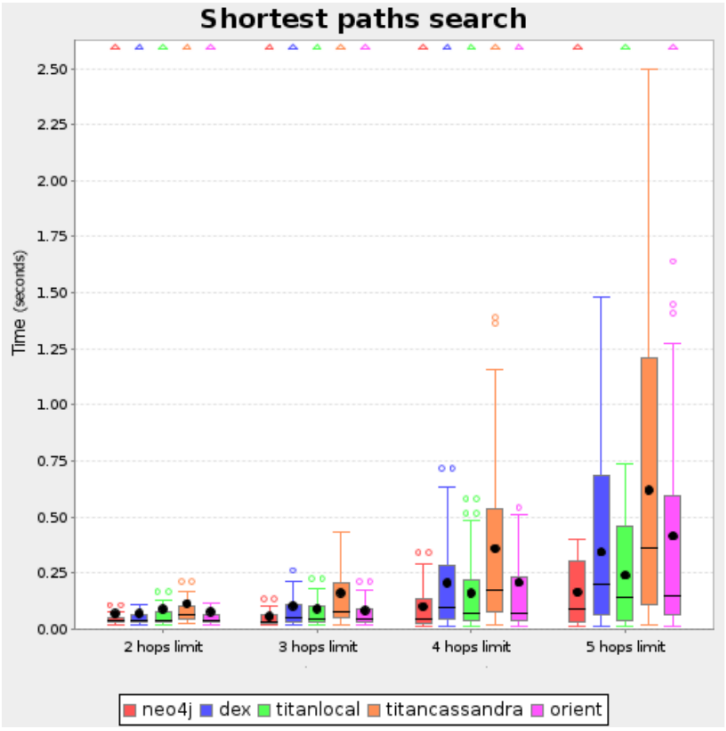

Figure 7.18: Reading and traversing the data entries (Jouili & Vansteenberghe, 2013).

The authors of *Neo4j in Action* (Vukotic, Watt, Abedrabbo, Fox, & Partner, 2015) compared *Neo4j* with a relational database. Their goal was to investigate the difference in performance levels. The test environment was that of a social network with various levels of friends. Each level represented a certain depth of friends, for example friends-of-friends-of-friends (Vukotic et al., 2015). The results indicate that Depth Level 2 of the test did not yield considerably different levels of performance from the relational database and *Neo4j*. However, a significant difference in reading performance can be

seen at Depth Level 3 (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015). The reading performance of *Neo4j* does not increase in large intervals as the depth level increases. Instead, it increases linearly with the depth level (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015). Figure 7.19 is a summary of the results.

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Figure 7.19: Reading speeds of *Neo4j* (Robinson, Webber & Eifrem, 2015; Vukotic et al., 2015).

Batra and Tyagi (2012) compared *Neo4j* with *MySQL*. Their goal was to investigate the performance differences between relational and non-relational data stores. Their test had various levels in which the database had to search for results. The execution times of the read requests are tabulated in Table 7.8. The results indicate that *Neo4j* completed the queries in less time than *MySQL* (Batra & Tyagi, 2012). To complete the largest query, *Neo4j* took 21 seconds, while *MySQL* took 620.56 seconds. This result is a good indication that *Neo4j* is formidable at reading results at multiple depth levels.

Table 7.8: Results of the performance tests of *Neo4j* and *MySQL* (Batra & Tyagi, 2012).

No_of_objects	MySQL:S0	Neo4j:S0	MySQL:S1	Neo4j:S1	MySQL:S2	Neo4j:S2
100	19.56	8	33	12.65	111.334	19.57
500	281.38	10	333.96	17	620.56	21

The results of these studies indicate that *Neo4j* can provide good read and adequate write performance, because *Neo4j* is read optimised. Its superior read performance becomes especially noticeable when working with larger datasets with more depth levels. The studies of Jouili and Vansteenberghe (2013), Robinson, Webber and Eifrem (2015), Vukotic et al. (2015), and Batra and Tyagi (2012) indicate that *Neo4j*'s read performance is superior, especially with heavily linked data. If a use case is very relationship driven or employs linked data, graph databases can be used for the use case. A grade of 7 for reads and grade of 5 for writes are assigned to *Neo4j*. This reflects the performance levels achieved for read and write requests in graph databases.

7.4.5 Scalability

Scalability refers to the system's ability to deal with increasing workloads (Orend, 2010). The *Neo4j* HA component allows the database to run in a clustered setup, meaning the database is distributed across several machines. This enables *Neo4j* to scale (Vukotic et al., 2015). A *Neo4j* cluster consists of one Master node and zero or more Slave nodes that all have full copies of the data (Neo4j, 2017). Figure 7.20 is a graphical representation of the replication process to scale the data among the cluster nodes. *Neo4j* employs Master/Master replication and all nodes can accept read and write requests (Hecht & Jablonski, 2011; Neo4j, 2017).

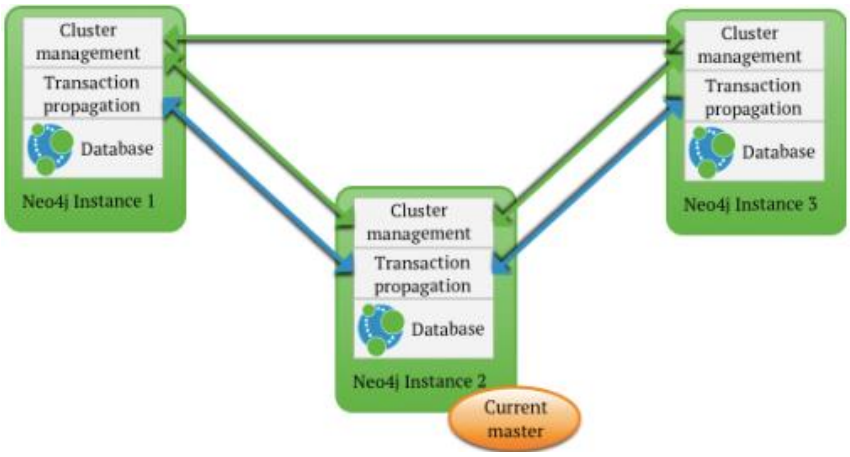

Figure 7.20: Graphical representation of replication between the Master and Slave instances (Neo4j, 2017).

Unlike the normal Master/Slave replication model, *Neo4j* can accept read and write requests through both Masters and Slaves (Vukotic et al., 2015). However, write requests are handled differently when received by a Slave than by a Master. A write request through a Master will be completed under normal conditions and the Master will update the dataset. A write request through a Slave will require a Master to be available to update the dataset (Vukotic et al., 2015). If a Master is down, an election process takes place during which a new Master is elected. This, along with the HA component, assists with the scalability of the cluster, because it ensures that a Master is always available.

Write operations are done through the Master, whereas read operations can be done locally on each Slave (Montag, 2013). This means that the read capacity of a cluster with the HA component will increase proportionally with the amount of running servers (Montag, 2013). If a cluster with five instances is serving five hundred read requests per second, the addition of a sixth instance would increase its capacity to six hundred read requests per second. Thus, the scalability performance level is also influenced by the number of running servers (Montag, 2013).

Robinson, Webber and Eifrem (2015, p. 169) state that a future goal is to be able to fully partition a graph database across multiple machines without interference from the client's application (Robinson, Webber & Eifrem, 2015). A benefit of this would be that read and write access could be

scaled horizontally (Robinson, Webber & Eifrem, 2015). The authors state that graph databases currently struggle to scale a graph dataset horizontally (Robinson, Webber & Eifrem, 2015). If a graph dataset is scaled horizontally, unpredictable query times may occur, because graph traversals could go across multiple machines (Robinson, Webber & Eifrem, 2015).

The information above leads to a grade of 5 being assigned to the graph stores' ability to scale. While it is possible to scale these datasets, performance will be unpredictable. The *Neo4j* HA component assists with replication and the performance of these replicas. A Master/Master replication method is employed and both Masters and Slaves can accept read and write requests. However, this method has the usual Master/Slave drawbacks. Focus is placed on the relationships between data. Therefore, if the graph dataset is partitioned among different servers, the performance could be unreliable due to traversals across the various nodes. Graph stores do not have a solution to both facilitate scalability and provide superior performance. However, replication can be seen as a method of providing scalability, because the different servers contain full copies of the graph data. Therefore, *Neo4j* is assigned a grade of 5 for scalability.

7.4.6 Conceptual data structure

Relationship-heavy data may be difficult to store in databases that do not focus on the relationships between the data. Graph stores place a focus on the relationships between data, while the other NoSQL families do not (Hecht & Jablonski, 2011). Examples of relationship-heavy data include social media data, transactional data, geospatial data, and linked data (Hecht & Jablonski, 2011). Social media data, for example data from Facebook or Twitter, is a good example of relationship-heavy data. A user can have friends and friends of friends (Robinson, Webber & Eifrem, 2015). The depth levels of relationships with friends and followers can become vast (Hecht & Jablonski, 2011). Graph stores, such as *Neo4j*, can handle these depth levels easily, while also providing high performance for queries (Robinson, Webber & Eifrem, 2015).

Robinson, Webber and Eifrem (2015) list some use cases in their book about *Neo4j*. The use cases depict how various data types can be utilised with graph stores to provide value (Robinson, Webber & Eifrem, 2015, p. 106). A business can employ social media data within a graph store to gain a competitive advantage. Recommendations regarding the next best product to sell can be made by investigating the relationship strengths between products. Graph stores can analyse geospatial data to determine the route or distance between two regions (Hecht & Jablonski, 2011). Graph stores can store network and data centre management data. A graph store can graphically depict the network performance to assist with troubleshooting. Graph stores can give insight into network deployment for future recommendations. The use cases above employ data with unique structures. Graph stores,

such as *Neo4j*, can store and work with such semi- and unstructured data and place focus on the relationships between the data.

Consequently, graph stores receive a grade of 9 for the conceptual data structure criterion. This represents the performance and ability of graph stores. Graph stores are the only NoSQL family that can provide superior performance when working with relationship-heavy data. Graph stores can store and work with semi- and unstructured data types. However, focus is placed on the relationships between datasets. The use cases mentioned some of the different data types that can be stored within graph stores. Graph stores may be the correct storage medium for a use case that employs relationship-heavy data.

7.4.7 Reliability

Reliability refers to the system's ability to operate without failures for a certain amount of time (Domaschka, Hauser & Erb, 2014). If a database is reliable, it may perform its function without any failure. Reliability in *Neo4j* is enabled by the HA component, which employs a Master/Slave replication model.

In a cluster with the HA component, the entire graph dataset is replicated to each instance in the cluster (Montag, 2013). The advantage of replication is that the data is safe even if a server fails. The disadvantage of replication is that it is resource intensive. The entire graph dataset needs to be able to fit into the capacity of every instance (Montag, 2013). The current version of *Neo4j* has no limitation on the number of nodes per instance it can store. Each *Neo4j* instance can store more than 34 billion nodes (Neo4j, 2017). Therefore, replication can be completed more easily, and better data reliability can be provided.

The *Neo4j* HA component requires a quorum to be present in the cluster to serve write requests (Neo4j, 2017; Montag, 2013). A quorum is reached when more than half of the nodes in the cluster are active. This allows elections to take place when a Master goes down so that write requests can be made (Neo4j, 2017). This is the method *Neo4j* uses to continue operating even after a failure occurs.

A grade of 8 is assigned to the reliability criterion for graph stores. A grade of 8 reflects the level of reliability that can be achieved within graph stores. Graph stores have certain drawbacks, such as Master/Slave elections and quorum requirements. Elections can result in data loss and down time. If a quorum is not present for an election in the *Neo4j* cluster, the cluster will degrade to read-only operation. This mode of operation allows only read requests to be completed until a quorum is established. Nevertheless, *Neo4j* is operational and allows requests to be made. The dataset is replicated to all the servers, which ensures the data is safe and reliable. Thus, graph stores can provide superior reliability.

7.4.8 Learning curve

Learning curve refers to the complexity level of a database as well as the time and effort needed to set it up and to learn how to use it. The focus of this criterion is on the available knowledge regarding graph stores and *Neo4j*.

Neo4j in Action by Vukotic et al. (2015) is a book for beginners who want to learn more about graph stores and *Neo4j*. *Neo4j in Action* contains 281 pages that explain graph stores and *Neo4j* in detail. The book starts by making a case for *Neo4j*. The book explains what *Neo4j* is and indicates how *Neo4j* and graph stores compare to other NoSQL stores. The book continues by explaining relevant terms and technologies used by *Neo4j*. Lastly, the book describes how *Neo4j* functions in a production environment. *Neo4j in Action* is aimed at novice IT practitioners.

The *Neo4j Cookbook* by Goel (2015) has 205 pages that focus on how to use *Neo4j*. The book explains single node setups as well as advanced setups with numerous technologies. Each example in the book is accompanied by code to assist the reader in learning to use and set up *Neo4j*. This book contains industry examples to illustrate how graph stores function in a contemporary technology context. The book covers advanced setups of *Neo4j* in distributed environments. It also investigates the scaling of *Neo4j* and graph stores. Lastly, the book explains how to do maintenance and administration for a *Neo4j* database. The *Neo4j Cookbook* (Goel, 2015) focusses on advanced usage of *Neo4j* and how *Neo4j* can fulfil various needs.

The *Neo4j* website (https://neo4j.com) contains a wide variety of resources. All the *Neo4j* documentation data of can be found on the website. Research papers on *Neo4j* can also be obtained from the website. On the website, a download link for *Neo4j* allows clients to download and install *Neo4j*. There are also numerous tutorials for beginners and advanced users on YouTube (www.youtube.com). Below is a list of popular books about *Neo4j*. The books can be used to train users to take advantage of graph stores and *Neo4j*.

- Van Bruggen, R. (2014). *Learning Neo4j: run blazing fast queries on complex graph datasets with the power of the Neo4j graph database*. Birmingham, UK: Packt Publishing.
- Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., & Partner, J. (2015). *Neo4j in Action*. Shelter Island, NY: Manning Publications Company.
- Gupta, S. (2015). *Neo4j essentials*. Birmingham, UK: Packt Publishing Limited.
- Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: new opportunities for connected data* (2nd ed.). Sebastopol, CA: O'Reilly Media, Incorporated.
- Goel, A. (2015). *Neo4j Cookbook: harness the power of Neo4j to perform complex data analysis over the course of 75 easy-to-follow recipes*. Birmingham, UK: Packt Publishing Ltd.
- Raj, S. (2015). *Neo4j high performance design, build, and administer scalable graph database systems for your applications using Neo4j*. Birmingham, UK: Packt Publishing.

- Redmond, E., Wilson, J. R., & Carter, J. (2012). *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Dallas, TX: Pragmatic Bookshelf.
- Lal, M. (2015). *Neo4j graph data modeling: design efficient and flexible databases by optimizing the power of Neo4j*. Birmingham, UK: Packt Publishing Limited.
- Jordan, G. (2014). *Practical Neo4j*. Berkeley, CA: Apress.
- Kemper, C. (2015). *Beginning Neo4j*. Berkeley, CA: Apress.
- Webber, J., & Robinson, I. (2016). *A programmatic introduction to Neo4j*. Harlow: Addison-Wesley.

Performing a search on the databases of Web of Science, IEEE Xplore, and ScienceDirect as well as on Google Scholar returned the following number of results. The keyword used for the search was "neo4j".

- Web of Science: 22 results
- IEEE Xplore: 36 results
- ScienceDirect: 107 results
- Google Scholar: 5030 results

As can be seen above, there are a several books available about graph databases, such as *Neo4j*. Some of the most popular books have been listed above. Graph databases focus on relationship-heavy data. This may require a different mentality when attempting to learn about a new graph database technology. The books range in content from the fundamentals of graph stores to experienced and advanced usage of *Neo4j*. There are many tutorials online that an individual follow to learn how to use *Neo4j*. *Neo4j* has a website (https://neo4j.com) that individuals can visit to download the software and read the latest documentation about *Neo4j* and graph databases. Graph stores are assigned a grade of 8 for the amount teaching materials available.

A grade of 8 means that individuals can teach themselves how to use graph stores, such as *Neo4j*. Books and tutorials on the subject are readily available. Beginners might struggle with graph databases, as they require individuals to think more about relationships. However, the amount of documentation available is still sufficient. According to the search results, not as much research is being done into graph databases as into the previous two NoSQL families. However, research is continually being done to improve graph stores, such as *Neo4j*.

7.5 Key-value stores (*Redis*)

The key-value data model stores data against a specific key. The data is stored as uninterpreted byte arrays, and the key is used to store, find, and sort the data (Abramova, Bernardino & Furtado, 2014; Hecht & Jablonski, 2011). The data is independently stored, which means that relationships between the data must be handled by the application logic (Hecht & Jablonski, 2011). The simple data structure of key-value stores enables schema-free storage. Any type of value can be added during runtime without affecting the availability of the database (Hecht & Jablonski, 2011). Key-value

stores prioritise scalability over consistency (Strauch, Sites & Kriha, 2011). Therefore, key-value stores may be able to address the need for access to distributed data.

Redis is an implementation of key-value databases. *Redis* stores data as values against keys. The keys are used to uniquely identify the data stored in *Redis*. *Redis* can provide fast access to the data because data is stored in main memory. *Redis* is schema-free and does not place any restrictions on the data it stores. The client application logic is required to process the value of and relationships between the data (Seguin, 2012, p. 7). Figure 7.21 shows key-value pairs that consist of different data values stored against their respective keys.

Key	Value
K1	AAA,BBB,CCC
K2	BBB,DDD
K3	PPP,KKK,123
K4	AAA,2,05/06/2006
K5	569,PIO,1234,ZXC

Figure 7.21: Key-value store contents, adapted from Wellhausen (2012).

7.5.1 Consistency

High consistency is achieved when clients always read the most up-to-date information (Pokorny, 2013). If high consistency is not supported, the most up-to-date information is not displayed. In time, the out-of-date information will be updated and the newest information displayed. This process is known as eventual or weak consistency (Pokorny, 2013). *Redis* employs eventual consistency due to the Master/Slave replication model (Hecht & Jablonski, 2011).

Redis uses a Master/Slave model to replicate data onto different nodes in an asynchronous manner, which means that the data is not replicated immediately (Das, 2015). The Master node will write all the data and then replicate the dataset to the Slaves. Therefore, the data on the Master and the Slave nodes is eventually consistent. An advantage of eventual consistency is increased performance. If high consistency is employed, any update or write to a Master must be replicated to the Slaves immediately. A large number of Slave nodes means that a large amount of resources will be used to update the data to all the Slaves (Das, 2015, p. 107).

The Master/Slave model also has drawbacks. An election process occurs when a Master node experiences a fault. When a new Master is elected, the Slave nodes must reconfigure to view the new Master. *Redis* employs a technology known as Redis Sentinel that aims to automate the reconfiguration of Slave nodes, which used to be a manual process (Da Silva & Tavares, 2015). Redis Sentinel aims to combat the drawbacks of the Master/Slave model by automatically promoting a Slave to the role of Master (Da Silva & Tavares, 2015). Redis Sentinel does not replicate data.

However, it ensures that the Master and Slave nodes are operational to provide consistency to the data (Da Silva & Tavares, 2015, p. 171).

Redis employs eventual consistency, which means that the Master dataset is not immediately replicated to the Slaves. The Slave nodes contain old copies of the data until an update operation occurs. The drawbacks of the Master/Slave model are mitigated through Redis Sentinel. The election process is automated to ensure a new Master is elected and the Slaves are reconfigured to view the new Master node. Therefore, key-value stores can provide adequate levels of consistency. A grade of 6 represents the level of consistency of data within key-value stores. A combination of eventual consistency and the Master/Slave model enables above-average levels of consistency to be achieved. *Redis* can provide asynchronous consistency of data with reliable performance benefits. Thus, if a use case does not require synchronous consistency of data, *Redis* can be employed to provide eventual consistency.

7.5.2 Availability

The percentage of time a system is functioning correctly is an indication of its availability (Orend, 2010). Availability also refers to continuous operation after a fault occurs (Han, Haihong, Le & Du, 2011; Strauch, Sites & Kriha, 2011). *Redis* employs the Master/Slave replication model to provide availability and fault tolerance (Das, 2015). This replication model allows the Master to replicate the dataset to the connected Slaves (Carlson, 2013). The updated Slaves make it possible for the clients to retrieve the full dataset by ensuring the data is available.

The availability of *Redis* depends on the availability techniques of the Master and Slave nodes. The availability technique used by Slaves differs from the Master's technique. There are numerous Slave nodes that can be used to counter a failure. The available Slave nodes will accommodate the failed Slave's requests to ensure that clients can access the dataset (Das, 2015). The technique employed by the Master node is more difficult because there is only one Master at a time (Das, 2015; Da Silva & Tavares, 2015). If the Master node fails, write requests will not execute until a new Master node is running (Das, 2015). This means that data loss can occur, because only read requests can be completed. A technology known as Redis Sentinel can be used to combat data loss (Das, 2015, p. 265). Redis Sentinel performs the election process automatically and configures all the Slaves to point to the new Master node (Da Silva & Tavares, 2015). Therefore, Redis Sentinel attempts to combat the downtime and data loss that can occur if a Master node fails (Da Silva & Tavares, 2015, p. 171).

Redis can provide high availability through the Master/Slave replication model and Redis Sentinel. The data is always safe, because each node contains a copy of the entire dataset. However, a

drawback of this replication method is experienced when a Master node fails, because write requests may not be executed without the presence of a Master node. Therefore, to prevent data loss, it is important that a Master node always be present. Redis Sentinel is the solution to the data loss problem. It automatically elects a new Master from among the Slaves to ensure that write requests can continue. High availability can be achieved with key-value stores. Therefore, a grade of 8 is assigned to the availability criterion. This reflects the level of availability key-value stores can provide if they are set up correctly.

7.5.3 Partitioning

There are limits to the capacity and performance of database servers. Exceeding these limits requires the database to be partitioned across multiple database clusters. NoSQL families' methods of distributing data to multiple machines differ (Hecht & Jablonski, 2011). The majority of the NoSQL families have a key-oriented data model. There are two methods that can be used to partition data. The first method is range-based partitioning, and the second method is consistent hashing (Redis, 2017).

The range-based partitioning strategy distributes a dataset by using the range of the dataset's keys (Hecht & Jablonski, 2011). The keyset is split into blocks and each block is stored on a different node (Chen, Mao & Liu, 2014). For example, users with an ID of 0 to 10 000 are stored on instance R0, and users with an ID of 10 001 to 20 000 are stored on instance R1 (Redis, 2017). Range-based partitioning allows efficient handling of queries, as the keys and their associated data are stored on the same node (Hecht & Jablonski, 2011). A disadvantage of this method is the table required to manage the mappings of ranges to instances (Redis, 2017). The table must be constantly updated to keep track of stored records. If the table were to fail, the queries could be sent to the wrong instances, which could cause downtime. Therefore, a single point of failure exists within the range-based method.

The consistent hashing method allows for higher availability (Hecht & Jablonski, 2011; Karger et al., 1999). This strategy employs a shared nothing architecture (Stonebraker, 1986) and does not have a single point of failure. The keys are distributed through hash functions to allow quick calculation of a key's address within the cluster (Hecht & Jablonski, 2011). Consistent hashing does not require a table to keep track of the data locations. However, since the keys are distributed at random throughout the cluster, this strategy may cause performance penalties on queries due to the high network load (Hecht & Jablonski, 2011). Hecht and Jablonski (2011) gave *Redis* a positive rating for consistent hashing and a negative rating for range-based partitioning. These ratings indicate that *Redis* should employ consistent hashing and not range based partitioning.

Key-value stores receive a grade of 6 for the partitioning criterion. The two methods of partitioning each have their own advantages and disadvantages. Range-based partitioning has a single point of failure but provides superior performance. Consistent hashing does not have a single point of failure. However, performance penalties may occur since a tracking table is absent. Therefore, it is possible to provide above-average performance through partitioning data in *Redis*. The advantages and disadvantages of the two methods must be investigated before implementing partitioning in *Redis*.

7.5.4 Read and write performance

In a study done by Du Toit (2016), the reading and writing capabilities of different NoSQL databases were evaluated. Workloads of various sizes were implemented and the time until completion was recorded. Du Toit (2016) performed bulk inserts of various amounts of records into several NoSQL databases. Du Toit (2016) found that *Redis* does not allow batch loading from a client application. The batch loading of data had to be done through a text file and a batch load program (Du Toit, 2016). Owing to this, Du Toit (2016) stated that its performance could not be tested by the benchmark tool. Du Toit (2016) added the total dataset as a single record through the benchmark tool and found the writing speed to be 51 records per second and 19.56 milliseconds per record (Du Toit, 2016).

The reading performance of *Redis* indicates that result sets of up to 20 000 records were possible (Du Toit, 2016). *Redis* could read a single record in an average time of 9.3 milliseconds (Du Toit, 2016). However, Table 7.9 indicates that a single record took longer than any batch reading query (Du Toit, 2016). The fastest time recorded was 3.26 milliseconds per record for a batch read job of a 1000 records (Du Toit, 2016). The read performance of *Redis* remained relatively consistent up to 20 000 records where there was a slight increase to 4.4 milliseconds (Du Toit, 2016). At its fastest, *Redis* could read 300 records per second when reading between 1000 and 5000 records in total (Du Toit, 2016). Its slowest performance of 227 records per second was recorded at 20 000 records (Du Toit, 2016). Du Toit's (2016) research indicates that *Redis* can provide superior read performance, especially during batch queries. Table 7.9 depicts the results of Du Toit's (2016) reading benchmark tests for *Redis*.

Table 7.9: Results of *Redis* reading benchmark tests over four nodes (Du Toit, 2016).

Records	1	100	1000	2500	5000	10000	20000
Duration in ms (average)	9.3	382.9	3258.9	8188.5	16644.9	34176.4	88079.1
Latency per record in ms	9.3	3.829	3.2589	3.2754	3.32898	3.41764	4.40396
Records per second	107.527	261.165	306.852	305.306	300.392	292.6	227.069

Abubakar, Adeyi and Auta (2014) compared the performance of *MongoDB*, *Redis*, *OrientDB*, and *ElasticSearch*. Figure 7.22 indicates that *Redis* outperformed the other NoSQL databases in the write benchmark test (Abubakar, Adeyi & Auta, 2014). *Redis* completed the write operation in the smallest amount of time (Abubakar, Adeyi & Auta, 2014). *Redis* provides superior write performance because the dataset is stored within main memory (Seguin, 2012).

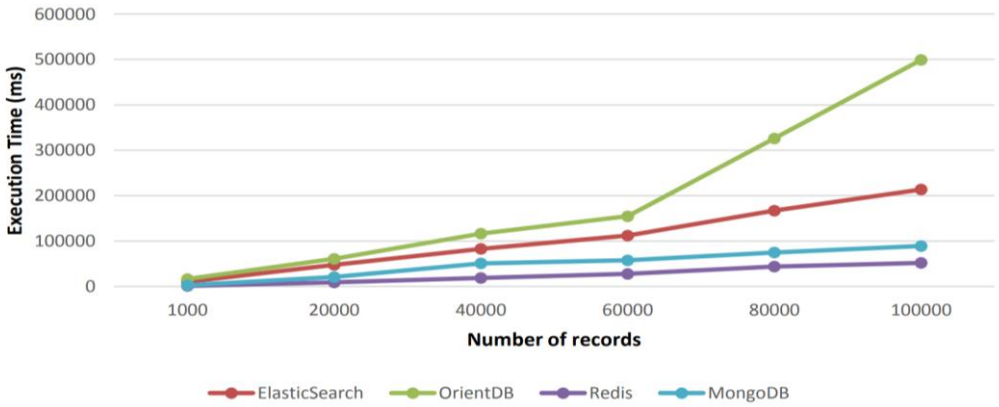

Figure 7.22: Write performance comparison (Abubakar, Adeyi & Auta, 2014).

Figure 7.23 indicates that *Redis* was the second best in the read benchmark test (Abubakar, Adeyi & Auta, 2014). Thus, according to the findings of this study, *Redis* provides superior read and decent write performance (Abubakar, Adeyi & Auta, 2014). This study's findings indicate that the read performance of *Redis* is superior to its write performance and thus agree with Du Toit's (2016) findings.

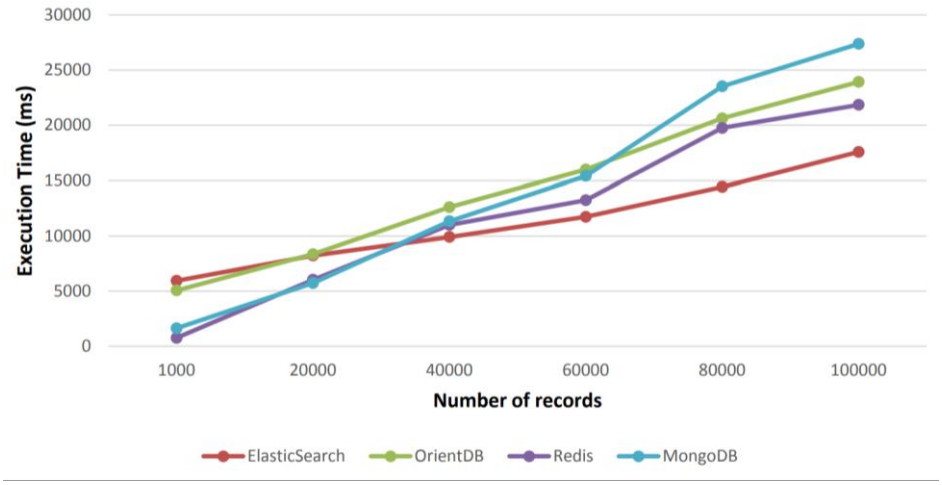

Figure 7.23: Read performance comparison (Abubakar, Adeyi & Auta, 2014).

Abramova, Bernardino and Furtado (2014) evaluated the performance capabilities of several NoSQL databases. The evaluation compared three of the four NoSQL families by comparing several NoSQL databases. The NoSQL databases represented their respective families. The aim of the evaluation

was to understand the effect of the data model on the performance of each family (Abramova, Bernardino & Furtado, 2014).

Figure 7.24 indicates the performance of each database in the write test. *Redis* was the fith fastest database to complete the operation (Abramova, Bernardino & Furtado, 2014). It took *Redis* 5 minutes and 17 seconds to write 600 000 records (Abramova, Bernardino & Furtado, 2014). The authors noted that *Redis* could not insert 600 000 records at once (Abramova, Bernardino & Furtado, 2014). Two write operations were used to write 600 000 records. The sum of the operation times was 5 minutes and 17 seconds (Abramova, Bernardino & Furtado, 2014).

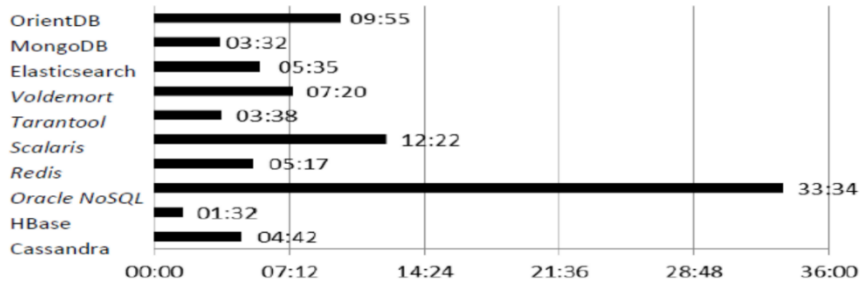

Figure 7.24: Write performance times (Abramova, Bernardino & Furtado, 2014).

Figure 7.25 indicates the performance of each database in the read test. *Redis* completed the test within 0.49 seconds, which was the second fastest time (Abramova, Bernardino & Furtado, 2014). The results of these tests show that the read performance of key-value stores is far superior to their write performance. Key-value stores use main memory to store data, which may be an indication of why their read performance is fast (Seguin, 2012).

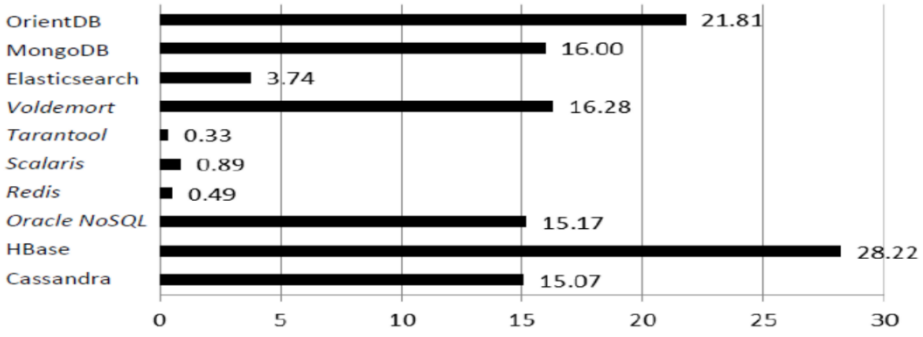

Figure 7.25: Read performance times (Abramova, Bernardino & Furtado, 2014).

Key-value stores can provide fast read and write performance, because data is stored in main memory, which makes it quickly accessible. Most of the performance studies above indicate that key-value stores provide superior reading and proficient writing performance. Therefore, grades of 7 and 6 are assigned to reading and writing performance, respectively. The grade of 7 means that

good reading performance is achievable within key-value stores, while the grade of 6 means that writing performance is above average. However, there are drawbacks to consider. The first drawback is that key-value stores use volatile memory to store datasets, which means that data loss can occur if power is lost. Another drawback is the capacity of main memory. The entire dataset is stored in main memory. Therefore, the size of the dataset is limited by the capacity of main memory. Large datasets may lead to a higher cost to accommodate the size of the dataset. Key-value stores are suitable for high-performance computing that employs moderately sized datasets.

7.5.5 Scalability

Scalability refers to the system's ability to handle increasing workloads (Orend, 2010). *Redis* achieves high scalability through replication and clustering (Redis, 2017). Replication of data is achieved through the Master/Slave model. Writes performed on the Master node are replicated to the Slave nodes (Redis, 2017). Thus, the Slave nodes have an exact copy of the Master node's data. Therefore, clients can query both the Master and Slave nodes (Carlson, 2013). The benefit of this replication model is that the data can be scaled to various nodes within the cluster. If failures were to occur, the data would be safe because exact copies can be found on all nodes (Da Silva & Tavares, 2015). The Master/Slave model does have drawbacks, such as downtime. However, Redis Sentinel aims to combat the drawbacks of the Master/Salve model (Da Silva & Tavares, 2015, p. 171).

Redis clustering is the second approach used to provide high scalability (Redis, 2017). *Redis* clustering is a method of automatically sharding the dataset across several nodes (Redis, 2017). Hash slots are employed to allow the addition or removal of nodes without the occurrence of downtime. Asynchronous replication exists between the nodes in the *Redis* cluster, meaning the Slave nodes are not updated immediately (Redis, 2017). The Master/Slave replication model ensures that data can be scaled. The drawback of the clustering method is that new nodes do not store data immediately. The cluster needs to be re-sharded to configure and add blocks to the new node (Redis, 2017). In doing so, data can be split across numerous *Redis* instances to provide high scalability.

The abovementioned leads to a grade of 7 being assigned to the scalability criterion. The grade of 7 reflects the high level of performance that key-value stores can provide where scaling is concerned. *Redis* employs two approaches to provide high scalability. The Master/Slave replication model replicates the dataset between the nodes. There are drawbacks to employing the Master/Slave model. However, Redis Sentinel automates the election process to mitigate these drawbacks. *Redis* clustering provides the ability to shard the dataset across numerous cluster nodes. This method allows new nodes to be added without downtime. However, if a new node is added, the cluster must be re-sharded to allocate blocks to store the data, meaning downtime could occur.

7.5.6 Conceptual data structure

Semi- and unstructured data can originate from various sources with different structures. Therefore, the storage technology must be able to handle the heterogeneous nature of the data. *Redis* is completely schema-free and allows the insertion of data during runtime (Hecht & Jablonski, 2011). Key-value stores are useful in use cases that deal with key-based attributes (Hecht & Jablonski, 2011). Common use cases include the management of user profiles, the management of sessions, and the retrieval of product names (Moniruzzaman & Hossain, 2013). *Redis* can support a rich set of data types that includes strings, hashes, lists, sets, and sorted sets (Redis, 2017). These five data types allow a range of problems to be solved (Carlson, 2013).

Some companies that use Redis include companies such as Twitter, Pinterest, Snapchat and Flickr among others (Redis, 2017). Redis can provide high performance while dealing with a rich set of data types. An example of a Redis use case is within the Twitter context where Redis is used to create an individual's timeline (Iravani, 2015). The timeline consists of tweets indexed by their id which allows Redis to chain the tweets together, regardless of the data type (Hecht & Jablonski, 2011; Iravani, 2015). As a result, Twitter employs Redis as it meets the performance and data structure requirements of its use case. Other companies such as Pinterest and GitHub have different data type requirements than that of Twitter (Shon, 2014). However, each of the companies employ Redis since it can accommodate their data type requirements. Therefore, key-value stores can accommodate various data type requirements.

Consequently, key-value stores receive a grade of 7 for the conceptual data structure criterion. This family is able to store and work with semi- and unstructured data. Key-value stores, such as *Redis*, can accommodate numerous data types. Therefore, they can be used to solve numerous problems.

7.5.7 Reliability

If the reliability of a database system is high, then the database system is less likely to fail (Domaschka, Hauser & Erb, 2014). A method of combatting weak reliability in key-value stores is persisting data to disk (Redis, 2017). Within *Redis*, there are two methods of persistence, known as snapshotting and append only file (AOF), which can be used separately, together, or not at all depending on the situation (Carlson, 2013; Redis, 2017). The choice between these methods depends on the data stored and the application implemented (Carlson, 2013). One of the primary reasons to use these methods is to provide high reliability of data. Key-value stores use main memory to store the data. Main memory is volatile, which means the data can be lost. Therefore, main memory data is persisted to disk to ensure the reliability of data (Carlson, 2013; Redis, 2017).

Snapshotting is a persistence method that writes the current data to a disk at specific intervals (Carlson, 2013). Snapshotting is appropriate when working with datasets less than a few gigabytes in size (Carlson, 2013; Redis, 2017). Snapshotting is a method used to persist data to disk (Carlson, 2013). As the dataset size increases, the time needed to persist the data increases. There are two drawbacks to implementing snapshotting. The first drawback is that a large dataset leads to less memory being available. Therefore, snapshotting will take more time to complete (Carlson, 2013). The performance of *Redis* can degrade heavily if snapshotting is slow to complete (Carlson, 2013). The second drawback is data loss. If a crash occurs, all the data modified since the last snapshot is lost (Carlson, 2013; Redis, 2017).

Append only file (AOF) is the second method used to provide high reliability. AOF enables *Redis* to store more up-to-date data (Carlson, 2013). The AOF log tracks all changes that occur in the dataset. Thus, the dataset can be recovered from the AOF log if faults occur (Carlson, 2013). The configuration of AOF can meet various needs, such as persisting data to disk every second. Such a configuration means that only one second of data will be lost if a failure occurs. A drawback to AOF is that increased storage capacity is used. AOF can also be slower than snapshotting the dataset (Redis, 2017).

A grade of 7 is assigned to key-value stores for the reliability criterion, because they employ different methods to ensure data reliability. Key-value stores can provide superior reliability through each method. Each method has its own drawbacks and advantages. However, the choice of method depends on the circumstances and the applications used.

7.5.8 Learning curve

The time and effort needed to set up and learn how to use a database as well as the database's complexity level are represented by the learning curve criterion. *Redis in Action* (Carlson, 2013) is a book that teaches the reader about *Redis* and key-value stores. The book contains 293 pages that introduce the reader to *Redis*. Many relevant terms associated with *Redis* and key-value stores are explained. The book also explains the data types that *Redis* can work with as well as how several problems can be solved through *Redis*. The last part of the book explains how *Redis* can scale within a distributed environment. This book is aimed at novice readers who want to get started with *Redis*.

Redis Essentials (Da Silva & Tavares, 2015) is a book of 197 pages that aims to teach readers how to use *Redis* in a business environment. The topics covered include the installation of *Redis* and how to implement the *Redis* cluster and Redis Sentinel technologies. The book includes some example code to assist the reader in understanding the techniques being implemented. The book is aimed at both beginners and advanced readers who want to employ *Redis*.

The *Redis* website (https://redis.io) supplies a wide variety of resources, including the most up-to-date *Redis* documentation. A full set of commands for *Redis* is available on the website. A download link for *Redis* is available for clients to download and install the database. There are also numerous tutorials for beginners and advanced users on YouTube (www.youtube.com). Numerous other learning materials can be found on the internet. These materials include research materials, tutorial, and books. Some popular books about *Redis* and key-value stores are listed below. These books provide the reader with the fundamentals of key-value stores. They can be used to teach users the most basic concepts and most advanced usage of *Redis*.

- Carlson, J. L. (2013). *Redis in Action*. Shelter Island: Manning.
- Macedo, T., & Oliveira, F. (2014). *Redis Cookbook.* Beijing; Köln; Sebastopol, Calif.: O'Reilly.
- Dayvson, D. S., & Tavares, H. L. (2015). *Redis Essentials*. Packt Publishing.
- Das, V. (2015). *Learning Redis*. Packt Publishing.
- Redmond, E., Wilson, J. R., & Carter, J. (2012). *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Dallas, TX: Pragmatic Bookshelf.
- Nelson, J. (2016). *Mastering redis*. Packt Publishing Limited.
- Chinnachamy, A. (2013). *Instant redis optimization how-to*. Packt Publishing Limited.
- Palmer, M. (2013). *Instant redis persistence*. Packt Publishing Limited.
- Sanfilippo, S., & Noordhuis, P. (2011). *Redis: The Definitive Guide Data Modeling, Caching, and Messaging*. Oreilly & Associates Inc.
- Chinnachamy, A. (2014). *Redis applied design patterns: use Redis' features to enhance your software development through a wide range of practical design patterns*. Birmingham: Packt Publishing.

Performing a search on the databases of Web of Science, IEEE Xplore, and ScienceDirect as well as on Google Scholar returned the following number of results. The keyword used for the search was “redis”.

- Web of Science: 21 results
- IEEE Xplore: 47 results
- ScienceDirect: 0 results
- Google Scholar: 105 000 results

However, the content received was not always in line with the keyword. For example, a change in search term from “redis” to “redis database” delivered only 13 500 results on Google Scholar.

There are various materials available that can be used to teach IT practitioners about key-value stores, such as *Redis*. The teaching materials consist of book tutorials, video tutorials, and courses about key-value stores. The material is readily available and *Redis*’s documentation is available on their website (https://redis.io). The documentation is always expanding as new information is constantly added to the current body of knowledge.

Considering the number of search results and the amount of teaching materials available, key-value stores are assigned a grade of 7. The grade of 7 means that an individual can download information, install the database, and teach themselves through online tutorials and books. There are also more advanced tutorials and books available that show individuals how to set up and use *Redis* to solve problems. The results of the search performed on the databases show that some research is being done in this area, but not as much as with the previous NoSQL database families.

7.6 Conclusion

Each NoSQL family has its own benefits. However, there are also disadvantages to using each of the families, which means that there is no single technology that can fulfil all possible requirements. Table 7.10 provides a summary of the grading done in this chapter.

This chapter set out to grade the four NoSQL families on their performance by using a fixed set of criteria. The four NoSQL families each have unique strengths and weaknesses. For example, graph stores excel at using relationship heavy data to solve problems. Document-stores allow a multitude of data types to be stored together in a single document. Key-value stores provide the most consistent performance but have a storage limitation. Column-family stores provide proficient writing and consistency performance when working with heterogeneous data. There is no single solution for all use cases. However, each NoSQL family can be a better choice for a specific use case than the other families, as the capabilities of the families differ.

This chapter performed Step 3 (grade according to the criteria) of the 6-step process model. It graded the performance of each family by using a fixed set of criteria. The next chapter will apply the framework to a specific use case and provide a recommendation regarding which NoSQL family to choose. Chapter 8 will perform Step 4 (weight the criteria), Step 5 (score the options), and Step 6 (recommend an option) of the 6-step process model. The goal of the next chapter is to continue to demonstrate the feasibility and utility of the framework in a real-world use case by completing Steps 4 to 6 of the process model within the context of storing NetFlow data in a NoSQL database.

Table 7.10: Summary of grades for each NoSQL family.

Criterion	**Column-family**	**Document-based**	**Graph**	**Key-value**
Consistency	(7) Master/Slave replication provides good consistency.	(7) Replica sets replication provides good consistency.	(6) Master/Slave replication provides eventual consistency.	(6) Master/Slave replication provides eventual consistency.
Availability	(5) Master/Slave combined with RegionServers provides average availability.	(5) Replica sets and fault tolerance. Rollbacks can cause data loss.	(9) Master/Slave and cluster management provide excellent availability.	(8) Redis Sentinel and Master/Slave provide very good availability.
Partitioning	(7) Range-based partitioning provides good performance.	(7) Range-based partitioning and autosharding provide good performance.	(4) Difficult to partition. Can be partitioned through cache sharding.	(6) Consistent hashing provides low performance and better fault tolerance.
Read performance	(4) Weak read performance.	(9) Excellent read performance.	(7) Good read performance.	(7) Good read performance.
Write performance	(8) Strong write performance.	(5) Average write performance.	(5) Average write performance.	(6) Above-average write performance.
Scalability	(7) HDFS and autosharding provide high scalability.	(6) Autosharding and Master/Slave provide decent scaling but are difficult to set up.	(5) Difficult to scale. Master/Slave replicates full copies to all nodes.	(7) Master/Slave and Redis Sentinel provide good scalability.
Conceptual data structure	(8) HDFS and flexible schema. No heavily linked data.	(7) Schemaless. Documents and collections.	(9) Heavily linked data. Can also store different types of data.	(7) Can store several types of data.
Reliability	(7) HDFS provides high reliability.	(7) Replica sets and automatic failover provide high reliability.	(8) *Neo4j* HA component provides good reliability.	(7) Snapshotting and append only file provide good reliability.
Learning curve	(8) Easy to learn. Many learning materials available. Not too difficult to implement.	(9) Easy to learn. Many learning materials available. Not difficult to implement.	(8) Easy to learn. Many learning materials available. Not too difficult to implement.	(7) Not too difficult to learn. Some learning materials available.

CHAPTER 8: NETFLOW USE CASE

In this dissertation, a framework that can be used to assist IT practitioners in making decisions regarding NoSQL technologies is proposed. The proposed framework presents a weighted decision model that can be tailored to assist with specific technological use cases by following a 6-step process. The previous chapter adapted the weighted decision model by grading the four NoSQL families using a fixed set of criteria for NoSQL technologies. The current chapter will further demonstrate the feasibility and utility of the framework in a real-world use case and complete the adaption process.

The next step (Step 4) of the 6-step process model is to allocate weights to the criteria. The requirements of the use case determine what weight is allocated to each criterion. Input regarding the relative importance of each criterion is obtained from IT practitioners who are familiar with the requirements of the use case. The goal is to indicate which of the criteria are important for the success of the use case. After the weights are assigned, calculations can be performed to compute the final score for each family (Step 5), which can then be used to make a recommendation (Step 6).

The use case context for this study is NetFlow. This chapter starts by explaining the NetFlow use case. Thereafter, the requirements of the use case that will affect the weight values assigned to the criteria are investigated. The instrument used to gather the weight values is described and the final scores of the NoSQL families are discussed. Lastly, the recommendation provided by the framework is discussed and an explanation of why that specific recommendation was made is given.

8.1 Use case

Most networks have some sort of network monitoring or reporting tool in place. Network monitoring tools are used to gain an understanding of a network's operation. NetFlow is a popular protocol that can be used to report on a network's operation or identify irregularities if normal functioning is not occurring (Cisco, 2012). In the use case focused on in this study, IT practitioners must decide which NoSQL family to use to store captured NetFlow data for decision-making purposes.

Normal operational use of NetFlow will continue. However, storing the data in a NoSQL database will allow it to be used for additional purposes, such as trend analysis, security analysis, and decision support. It is essential to obtain weight values to indicate which criteria are important for this use case.

8.1.1 NetFlow

There are a variety of factors that can influence the amount and type of traffic that is generated inside a network (Sommer & Feldmann, 2002). Examples of such factors include a change in users'

behaviour on the network, the installation of new technologies or hardware, and the time of the day (Sommer & Feldmann, 2002). In order to make sense of the increase in and type of traffic on the network, the IT practitioner needs to have a clear understanding of what is occurring inside the network. Network measurement systems can provide aggregated information for each pair of IP addresses or port numbers (Sperotto et al., 2010). To gain an understanding of the network and its performance, the information regarding communications inside the network needs to be inspected. NetFlow is a tool that can be used to aggregate such information.

According to Cisco (2012), NetFlow is an instrument in Cisco IOS software that can be used to characterise the operation of the network. NetFlow data is network traffic data between two hosts. NetFlow data consists of NetFlow records. Each record contains information, such as IP addresses, port numbers, traffic protocol types, and the volume of traffic being sent across the network (Table 8.1) (Lakkaraju, Yurcik, & Lee, 2004). A prominent characteristic of NetFlow data is the rate at which the data is generated (Zhou, Petrovic, Eskridge & Carvhalo, 2014). An organisation with a major internet backbone can generate large volumes of NetFlow data at a high velocity. For example, routers of a university can generate large amounts of NetFlow data (Zhou et al., 2014).

Table 8.1: NetFlow fields and their meanings (Sommer & Feldmann, 2002)

Name	Description
Srcaddr	Source address
Dstaddr	Destination address
Input	Input interface
Output	Output interface
dPkts	Number of packets
dOctets	Number of octets
First	Start of NetFlow
Last	End of NetFlow
Srcport	Source port
Dstport	Destination port
Tcp_flags	TCP flags
tos	IP type of service

Capturing NetFlow data requires large amounts of storage space. The next section describes how NetFlow can add value to organisations.

8.1.2 The value of NetFlow

Storing NetFlow data without using it wastes resources. There are numerous ways through which NetFlow can add value to an organisation, such as intrusion detection, network monitoring, bandwidth estimation, and predicting future data requirements. This data can also be used to determine more in-depth information, such as the relationships between NetFlow data from several locations, future network requirements, and the cybersecurity state of the network, as well as to gain knowledge of the network. NetFlow data can be used to detect network intrusions (Sperotto et al., 2010), to estimate bandwidth for a network (Schmidt, Sperotto, Sadre & Pras, 2012), and for

visualising the data to solve problems (Minarik & Dymacek, 2008). Therefore, there are a variety of ways NetFlow can add value to an orginisation. However, the organisation may be required to be store the NetFlow data in a NoSQL technology. If so, a decision will have to be made regarding which NoSQL family to use to store the NetFlow data. An instrument will enable the IT practitioners to assign weights to criteria, which will assist with the decision-making process.

8.2 The instrument used to weight the criteria

In Chapter 6, three methods that can be used to capture the criteria weights from experts were investigated (section 6.2.2). These methods were interviews, focus groups, and questionnnaires.

Interviews offer the ability to examine experts' behaviour and answers to technical questions. However, the researcher did not have physical access to experts. Therefore, interviews were not feasible.

Focus groups enable a group of experts to voice their opinions and discuss the weight values in the context of a use case. However, since the researcher did not have access to co-located experts, a focus group could not be conducted.

For this study, a close-ended questionnaire with LPC-like scales is used. This enables the practitioners to enter specific weight values for each of the criteria. As stated before, the weights represent the level of importance of each criterion for the NetFlow use case. The questionnaire displays each criterion along with a short definition, which could assist the IT practitioner in assigning the correct weight. Refer to Appendix B for the instrument used in this study.

The method used has two rules for assigning weights. The first rule is that *each criterion can be assigned a weight of 1 to 10*. A scale of 1 to 10 is used to allow the IT practitioners to assign a value that is representative of the importance level of each criterion. Also, the scale can help prevent some decision-making biases from influencing the decision.

The second rule is that *a maximum of 50 marks can be assigned per questionnaire.* Placing a limit on a number of marks combats the effects of certain decision-making biases. Allowing a maximum of 50 marks to be distributed prevents individuals from assigning the same weight value to each of the criteria. It forces the IT practitioners to apply their minds to the task. Thus, this limit ensures that the recommendation provided is of a high quality.

Once the weights are assigned, the instrument determines whether more than 50 marks were assigned in total. If more than 50 marks were assigned, an error message is shown to indicate that the rules

were not followed. Thereafter, the instrument closes and records the answer. However, less than 50 marks may be assigned if it is of the IT expert's opinion.

Figure 8.1 is an example of the instrument that allows IT practitioners to assign weight values. Each criterion is represented on the left side as shown in Figure 8.1. Along with each of the criteria, a short definition is provided to assist IT practitioners in understanding the goal of each criterion. Both the minimum and maximum value that can be assigned are displayed to remind the IT practitioner of the rules. The IT practitioner is asked to assign the appropriate weight value to each of the criteria.

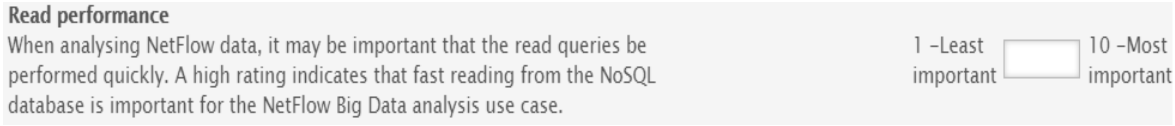

Figure 8.1: Example of the read performance criterion within the instrument.

The weighting instrument was distributed to expert IT practitioners who were asked to determine the importance of each criterion for the NetFlow use case. The identified weight values for the NetFlow use case will be inserted into the framework, which will recommend which NoSQL family to use.

8.3 Weight the criteria (Step 4)

The instrument that was developed to enable IT practitioners to insert weights was given to SANReN network engineers. The instrument requested that the IT practitioners indicate the importance of each criterion with regards to the NetFlow use case.

Additional proof that technology decisions are difficult to make was provided when the SANReN engineers were asked to complete the questionnaire. Some SANReN engineers felt that they did not have the required expertise to complete the questionnaire. This supports the claim that technology decisions are difficult to make.

Since such a decision is difficult, more than one opinion is necessary to derive an appropriate set of weight values. Therefore, a single opinion regarding the requirements of a use case is not a reliable source of information. More than one individual should complete the questionnaire from which the criteria weightings are derived. Since some SANReN engineers felt they did not have the required expertise, the questionnaire had to be distributed to other IT experts. Therefore, the questionnaire was distributed to NfSen and NFDUMP communities, whose members also have various levels of expertise in dealing with NetFlow data.

A total of six responses were received. However, only three responses were selected to be used. The final weightings for the NetFlow use case were derived from a comparison of these responses. Out of the six respondents, three respondents did not follow the rules of the questionnaire. One of the

respondents assigned 15 marks to one criterion, therefore the respondent did not follow the rules. Two respondents did not apply their minds to the task. They assigned many of the criteria a weight value of 10 and no values to others. This means that these two respondents did not carefully consider the importance of each criterion in the context of the use case to assign an appropriate weight value. The three remaining responses were deemed proper responses and used to determine the weights of the criteria. A high weight value indicates that it is important to achieve high performance within the specific criterion and a low weight value indicates that it is not important to have high performance within the specific criterion. The following sections will discuss each criterion by comparing the weight values assigned to it by the respondents.

Consistency

For the consistency criterion, the first respondent provided a weight value of 3, indicating that high consistency is not very important for the use case. The second respondent assigned a weight value of 6 to the criterion, which means that high consistency is of moderate importance to the use case. The third respondent assigned consistency a weight value of 8, indicating that high consistency is very important to the use case.

Two of the three weight values are relatively close to each other and indicate that consistency is important for the use case. However, the first respondent's weighting indicates otherwise. Owing to the large gap between the values, the average of the values may not properly represent the responses of the first and third respondents (section 2.2.1).

However, two of the respondents indicated that consistency is important. Therefore, the final weight assigned to consistency is 6, which represents the moderate importance of high consistency within the NetFlow use case. A weight of 6 means that the NoSQL family should provide above-average levels of performance where consistency is concerned.

Availability

Respondents 1 and 2 assigned a weight value of 5 to the availability criterion, while Respondent 3 assigned a weight value of 7. A weight value of 5 indicates that high availability is of average importance for the use case. However, a weight value of 7 indicates that high availability is very important for the use case. These values are relatively close to one another. Therefore, an average value can be used to represent the importance of the availability criterion. The final weight assigned is 6, which indicates that high availability is moderately important for the use case. Therefore, the NoSQL family should be able to provide above-average levels of performance where availability is concerned.

Partitioning

The first respondent assigned a weight value of 4 to the partitioning criterion. The second respondent assigned a weight value of 5, and the third respondent assigned a weight value of 7. A weight value of 4 indicates that a high level of performance is not very important where partitioning is concerned. A weight value of 5 indicates that it is of average importance, while a weight value of 7 indicates a high importance level.

There is a significant gap between the highest value of 7 and the lowest value of 4. Therefore, the average of the values would not be able to accurately represent the three respondents' opinions. However, two of the three respondents (Respondent 1 and 2) assigned weight values that are close to each other. This may be a better indication of the importance of the partitioning criterion for the use case. The final weight assigned to the partitioning criterion is 5. This indicates that it is of average importance to the use case. The NoSQL family should be able to provide average performance where partitioning and partition tolerance are concerned.

Read and write performance

The read and write criteria are weighted as two separate criteria. The read criterion will be investigated first. Respondent 1 assigned a weight value of 9 to the read criterion, indicating that this criterion is extremely important for the use case. Respondent 2 assigned a value of 8, indicating that this criterion is very important for the use case. Respondent 3 assigned a weight of 10, indicating that this criterion is vital for the use case. These grades are close to one another and are all very high grades. Therefore, it can be assumed that the reading performance of the NoSQL family should be very high to be able to fulfil the reading requirements of the NetFlow use case.

Respondent 1 assigned a weight value of 7 to the write criterion, which indicates that fast writing speeds must be achievable. However, Respondents 2 and 3 indicated that average writing speeds are adequate for the NetFlow use case by each assigning a weight of 5.

As the weight values assigned to both criteria are close to one another, their averages can be calculated to obtain final weight values to represent the read and write requirements of the use case. A final weight value of 9 is assigned to the reading criterion, and a final weight value of 6 is assigned to the writing criterion. The weight value of 9 means that high reading performance is extremely important, while the weight value of 6 means that high writing performance is only moderately important.

Scalability

Respondent 1 assigned a weight of 7 to the scalability criterion, which indicates that high scalability is very important for the NetFlow use case. Respondent 2 indicated that high scalability is moderately important by assigning value of 6, while Respondent 3 indicated that it is of average importance by assigning a value of 5. The highest value of 7 and the lowest value of 5 are relatively close to each other. Therefore, the average of the three responses can be used to indicate the importance of high scalability to the NetFlow use case. The final weight of 6 indicates that it is moderately important to provide high scalability within the NetFlow use case. Therefore, the NoSQL family should be able to provide above-average levels of performance where scalability is concerned.

Conceptual data structure

For the conceptual data structure criterion, the first respondent provided a weight value of 6, indicating that it is of moderate importance for the use case. Respondent 2 assigned a weight value of 7 to the criterion, indicating that it is very important for the use case. Respondent 3 provided a weight value of 1, indicating that the conceptual data structure criterion is of very low importance for the NetFlow use case.

The weight value of Respondent 3 contradicts the values of Respondents 1 and 2. This implies that there are contradicting opinions regarding the importance of the conceptual data structure criterion within the NetFlow use case. As there is a large gap between the values, the average of the three responses will not represent the view of Respondent 3. Therefore, the average of the three responses will not be used.

A final weight value of 4 is assigned to the conceptual data structure criterion. The value of 4 better represents the importance of this criterion within the NetFlow use case. All four of the NoSQL families can accommodate semi-structured data. This means that the NetFlow data can be stored and used with ease. Therefore, the weight value of 4 represents the less important nature of this criterion for this specific use case.

Reliability

For the reliability criterion, the first respondent provided a weight value of 3, indicating that high reliability is of low importance. The second and third respondents each assigned a weight value of 5 to the criterion, which means that high reliability is of average importance to the use case.

The highest and lowest value are close to each other. Therefore, an average value can be used to represent the importance of the availability criterion. The final weight assigned is 4, which indicates

that high availability is not very important for the NetFlow use case. Therefore, the database does not have to provide high levels of reliability.

Learning curve

For the learning curve criterion, the first respondent provided a weight value of 6, indicating that the learning curve criterion is moderately important for the NetFlow use case. The second respondent provided a weight value of 3, which means that the learning curve is of low importance for the use case. The third respondent agreed by assigning a weight value of 2, also indicating that the learning curve is of low importance for the use case.

There is a large gap between the highest and lowest value, which implies that there are different opinions regarding the importance of this criterion. Two of the three weight values are close to each other and indicate that the learning curve is not very important for the use case. However, the first respondent's weighting indicates otherwise. Owing to the gap between the values, the average of the values may not properly represent the first and third respondents' responses.

However, because two of the three respondents indicated that the learning curve is not very important, it can be assumed that the learning curve does not heavily affect the use case. Therefore, the final weight assigned to the criterion is 4. This indicates that the learning curve is not very important within the NetFlow use case.

Table 8.2 is a summary of the weights provided by the three respondents. It also indicates the final weights used to represent the use case requirements. The requirements need to be inserted into the model before it can provide the recommendation.

Table 8.2: Summary of respondents' and final weights for the model.

Criteria	Respondent 1	Respondent 2	Respondent 3	Final weight
Consistency	3	6	8	6
Availability	5	5	7	6
Partitioning	4	5	7	5
Read	9	8	10	9
Write	7	5	5	6
Scalability	7	6	5	6
Conceptual data structure	6	7	1	4
Reliability	3	5	5	4
Learning curve	6	3	2	4

8.4 Score the options (Step 5)

The final weights derived from the instrument can now be entered into the model to calculate the final score of each NoSQL family. The final scores are derived from the mathematical calculation represented below.

$$Score(F_k) = \sum_{i=1}^{n} W_i . R_{ik}$$

This calculation will provide each family with a final score. The higher the score, the more the family meets the use case requirements. Table 8.3 is a representation of the model and the final scores calculated for the families.

Table 8.3: Final scores of the NoSQL families.

Criteria	Weight	Column-family	Document-based	Graph	Key-value
Consistency	6	7	7	6	6
Availability	6	5	5	9	8
Partitioning	5	7	7	4	6
Read	9	4	9	7	7
Write	6	8	5	5	6
Scalability	6	7	6	5	7
Conceptual data structure	4	8	7	9	7
Reliability	4	7	7	8	7
Learning curve	4	8	9	8	7
Final score		325	341	333	339

The final score for column-family stores is 325. Document-based stores scored 341, while graph stores scored 333, and key-value stores scored 339. The highest score belongs to document-based stores, followed by key-value and graph stores, while column-family stores scored the lowest.

8.5 Recommend an option (Step 6)

The use case requires above-average levels of consistency. Document-based stores can provide high levels of consistency through their replica sets which replicate data. Therefore, they fulfil the use case's requirement regarding consistency. Column-family stores provide good consistency through Master/Slave replication, while key-value and graph stores provide eventual consistency.

The use case requires the database system to be moderately available at all times. Document-based stores can provide average availability through replica sets. Therefore, the family will perform adequately even though its performance regarding availability is slightly under what is required. As a result, document-based stores can satisfy the availability requirement of the use case. However, graph stores provide excellent availability through Master/Slave replication, while key-value stores

provide good availability through Master/Slave replication. This implies that graph and key-value stores can better satisfy the use case's availability requirement. Column-family stores provide only average availability through Master/Slave replication, therefore it can also meet the availability requirement.

The use case requires average performance where partitioning and partition tolerance are concerned. Document-based stores can provide high partitioning performance and be highly partition tolerant through range-based partitioning and autosharding. Therefore, they meet the use case's requirement regarding partitioning. Column-family stores employ range-based partitioning, and key-value stores employ consistent hashing, which means that these families can also fulfil the partitioning requirement of the use case. However, graph stores do not perform sufficiently where the partitioning criterion is concerned, since graph data is difficult to partition.

The most important requirement for the NetFlow use case is high reading performance. Document-based stores provide excellent reading performance. Therefore, they fulfil the most important requirement for the NetFlow use case. Column-family stores provide weak reading performance, while graph stores and key-value stores provide good reading performance. Therefore, they do not adequately meet the requirement of high reading performance.

The use case requires above-average writing performance. Document-based stores provide average writing performance, which could still satisfy the writing performance requirement. Document-based stores will perform adequately but not the best for the writing criterion. However, column-family and key-value stores are better equipped to provide the writing performance required by the use case. Graph stores also provide average writing performance, meaning it could also satisfy the writing requirement of the use case.

The use case requires above-average levels of performance where scalability is concerned. Document-based stores can provide above-average scaling through autosharding and Master/Slave replication. Therefore, they meet the requirement of the use case. Column-family stores employ the Hadoop Distributed File System (HDFS) with autosharding to provide good scalability performance. Key-value stores can provide good scalability performance through Master/Slave replication. Therefore, column-family and key-value stores could better satisfy the requirement of the use case. However, graph stores do not provide good scalability performance, because it is difficult to scale graph datasets. Therefore, graph stores do not meet the scalability requirement.

NetFlow is semi-structured data. The storage medium for the use case is required to be able to work with such data. All NoSQL families are able to meet the data structure requirement of the NetFlow

use case. Therefore, the conceptual data structure has an average level of importance within the use case.

The NetFlow use case does not require high levels of reliability. Document-based stores provide good reliability through Replica Sets and automatic failover. Therefore, they meet the use case's reliability requirement. Column-family stores use the HDFS (Hadoop distributed file system) to provide high reliability, while graph stores use the HA component to provide very good reliability. Key-value stores use snapshotting and the append-only file (AOF) methods to provide good reliability. Therefore all NoSQL families can provide the required reliability performance levels for the use case.

The learning curve is not very important for the NetFlow use case. Therefore, all of the NoSQL families are able to fulfil the learning curve requirement. There are many teaching materials available about all of the NoSQL families.

As can be seen in Table 8.3, the document-based family provides the best performance for the NetFlow use case. The document-based family fulfils the majority of its requirements, which were provided by the experts. Therefore, the recommendation from the model is to employ document-based stores within the NetFlow use case. Key-value stores receive an honourable mention, as their performance levels are close to those of document-based stores. Therefore, key-value stores could also be used within the NetFlow use case. However, a drawback to using key-value stores is that main memory is employed to store the dataset. This means that the size of the store's memory limits the size of the data set. NetFlow use cases can comprise of very large data sets to be stored. Therefore, the key-value store's memory may not be able to accommodate the size aspect connected to the NetFlow use case.

8.6 Conclusion

This chapter served as an instantiation to demonstrate the feasibility and utility of the framework. The decision framework was tailored to the NetFlow use case and used to recommend which NoSQL family to choose when storing NetFlow data in a NoSQL database. IT practitioners indicated the importance level of each criterion in terms of the use case through a data collection instrument. This made it clear that the criteria have various degrees of importance in the context of the NetFlow use case.

The weight values assigned to the criteria are representative of the use case requirements. This increased the quality of the recommendation provided by the framework. Once the weight values were inserted into the framework, the final scores could be calculated, and a recommendation could be made.

The recommendation was to use the document-based NoSQL family to store the NetFlow data. According to the framework, the document-based family is the best fit in terms of the requirements of the use case. The framework assisted the IT practitioner in making a more informed decision regarding which NoSQL family to choose. This demonstrated the utility value of the proposed framework.

PART D

EPILOGUE

CHAPTER 9: CONCLUSION

Decision-making is a difficult task and there are problems, such as biases and measurements, that can influence the outcomes of decisions. Therefore, there is a need for a framework that IT practitioners can employ to make better decisions. This study set out to provide a framework that can help IT practitioners make better decisions regarding NoSQL families.

The focus of this chapter is to revisit each sub-objective to indicate how the primary research objective was met. Each chapter within this study will be examined and explained as it pertains to the research objectives of this study.

9.1 Overview of the study

Decision-making is a core process of daily life (Nooraie, 2012). It was introduced in Chapter 1 as a major focus of this study. Chapter 1 introduced the problem area for this study as technology decision-making. The problem statement for this study was: *IT practitioners do not have a systematic way to select a NoSQL family for non-arbitrary use cases*. The problem statement indicated that IT practitioners struggle to make decisions regarding NoSQL technologies.

Design science influences this study. Therefore, the framework of March and Smith (1995) was employed to provide structure to the research. The study was broken into four parts. Part A focussed on context and discussed decision-making, biases, and NoSQL. Part B discussed the proposed framework, which consists of constructs, a decision model, and a process model in the form of the 6-steps. Part C focussed on instantiation in the context of a NetFlow use case. Part D is an epilogue, which is provided in the current chapter. These parts map well to the research framework of March and Smith (1995) as can be seen in Figure 9.1.

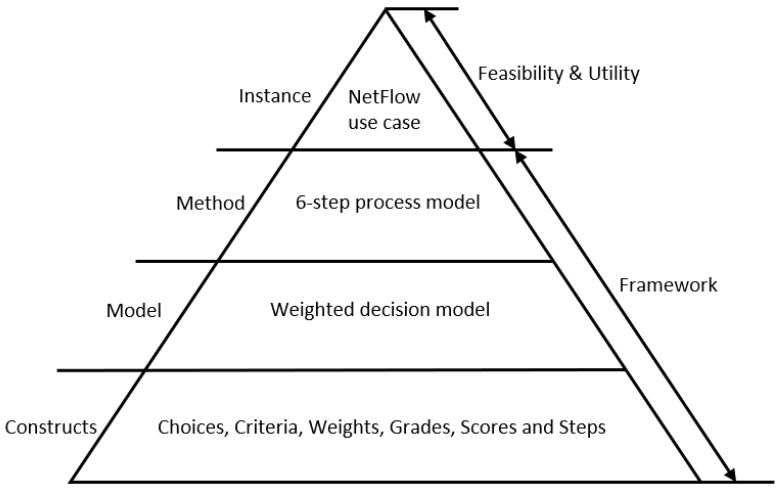

Figure 9.1: Overview of framework.

Chapter 2 introduced the context of the study and investigated decision-making and biases. The investigation showed that biases can negatively impact decisions. There are a wide variety of biases that can be classified using four categories that represent the overarching problems found in the biases. The biases focussed on in this study were grouped accordingly.

Chapter 3 added to the background of this study by investigating NoSQL storage technologies. This represented Step 1 of the 6-step process model. The focus of Chapter 3's investigation was NoSQL and the different NoSQL technologies. It was found that NoSQL has four families and many products associated with each family.

Chapter 4's goal was to provide an overview of the proposed framework. Chapter 4 discussed the need for a framework as well as how it should assist IT practitioners. Thereafter, a framework comprised of constructs, a weighted decision model, and a process model was proposed.

The decision-model combines all the constructs (options, criteria, grades, weights) through a final score and depicts their relationships with one another. The list of technologies is graded according to a fixed set of criteria to ensure all the options are properly investigated. Thereafter, the criteria are assigned weight values that represent the importance of each criterion within a use case context. The model calculates weighted final scores for the technologies to help the IT practitioner make an informed decision.

A specific method in the form of a process model is used to implement the decision model. The process model has 6 steps that IT practitioners can follow systematically to implement the framework. Each of the 6 steps plays an integral part in technology decision-making by mitigating the effects of measurements and biases. The 6-step process can also be used to adapt the framework to other contexts.

Chapter 5 focused on Step 2 of the 6-step process model, which is to identify comparison criteria. Therefore, the goal of Chapter 5 was to identify a fixed set of criteria that can be used to uniformly compare the NoSQL families. Nine criteria were identified, including the CAP criteria. A fixed set of criteria enables the IT practitioner to compare the NoSQL families uniformly.

Chapter 6 focused on explaining Steps 3, 4, and 5 of the 6-step process model. Chapter 6 started by explaining how to grade the options using the fixed set of criteria to ensure that all the options are properly investigated. Thereafter, the weights of the criteria were discussed. The weight values represent the importance of each criterion within a specific context. Lastly, the final weighted scores of the options were investigated and the calculation of the final scores through the mathematical formula was explained.

The above-mentioned chapters showed that the proposed framework can assist with technology decision-making. However, its feasibility and utility could still be questioned. Therefore, the framework was instantiated. Instantiation refers to placing an artefact within a specific instance to verify and demonstrate its use. The demonstration started in Chapter 7. The NoSQL families were graded according to the fixed set of criteria. This ensured that a uniform comparison of the NoSQL families, which would indicate their unique strengths and weaknesses, could be made.

Chapter 8 focused on the weighting of criteria within the NetFlow use case context to indicate the importance of each criterion for the use case. Once weights were assigned to the criteria, the final weighted score of each family could be calculated, which led to a recommendation of which technology to choose. The framework recommended the document-based family for the NetFlow use case, as it best fulfils the requirements. This recommendation was discussed and justified within the chapter. Chapter 8 also represented the final step of the 6-step process model.

9.2 Meeting the objectives

This section will look at how the objectives of this study were met.

The problem statement indicated the need for a systematic approach that IT practitioners can follow to make better decisions. Therefore, the primary research objective of this study was to *create a framework to help IT practitioners with NoSQL decisions*. To develop such a framework, various sub-objectives had to be met. The researcher first needed to understand decision-making and the problems that can influence decision-making. These problems also indicate why decision-making can be a difficult task. The study focused on technology decision-making and the researcher aimed to list typical problems that IT practitioners are faced with when making technology decisions. Thereafter, the researcher needed to identify a general model that can assist with decision-making and counter these problems. The model also needs the ability to adapt to specific scenarios.

To achieve the primary objective of developing a framework, three research sub-objectives needed to be addressed.

9.2.1 Enumerate typical decision-making problems

Sub-objective 1 (SO1) was to enumerate typical decision-making problems IT practitioners face when choosing between technologies.

The first sub-objective (SO1) was achieved in Chapters 2 and 3 (Part A) through a literature survey on decision-making, biases, and NoSQL. The problems and their effects on decision-making were investigated within the context of this study.

9.2.2 Identify a general model for decision-making

Sub-objective 2 (SO2) was to identify a general model for decision-making.

Sub-objective 2 (SO2) was accomplished through a literature survey done to find an appropriate framework as well as a mathematical expression with which to depict the framework. The framework was proposed and discussed in Chapter 4 (Part B).

9.2.3 Create a process to tailor the approach to the NoSQL scenario

Sub-objective 3 (SO3) was to create a process to tailor the approach to the NoSQL scenario. The context of this study was technology decision-making. It focussed specifically on decisions regarding NoSQL technologies. Therefore, a process to adapt the framework to the NoSQL scenario was proposed.

Sub-objective 3 (SO3) was partially met through a literature survey done to expand on the constructs within this study and a 6-step process model for adapting the framework to specific scenarios that was proposed through argumentation. This was discussed in Chapter 4.

Chapters 5 and 6 (Part B) also addressed the third sub-objective (SO3) through a literature survey and argumentation. The specific steps within the process model were discussed to indicate how to adapt the approach to the NoSQL scenario.

9.2.4 Create a framework

The researcher investigated decision-making and identified the problems faced when making decisions in Chapter 2. Since decision-making in general is a broad subject, a narrower context was found in technology decision-making and NoSQL, which were investigated in Chapter 3. Therefore, the first sub-objective was met. Next, the researcher needed to identify a model for making decisions. A framework to assist IT practitioners in making decisions regarding technology was proposed in Chapter 4. Therefore, the second sub-objective was met. A process model that can be used to adapt the framework to a specific NoSQL scenario was also proposed. The framework was discussed in detail in Chapters 5 and 6. Therefore, the third sub-objective was met.

As all three the sub-objectives were met, the main objective of this study, to *create a framework to help IT practitioners with NoSQL decisions*, was also met. However, the feasibility and utility of the framework could still be questioned. Therefore, instantiation was used to demonstrate the feasibility and utility of the framework within an instance. A NetFlow use case was used to demonstrate the utility of the framework in Chapters 7 and 8 (Part C). The framework provided the researcher with a recommendation of which NoSQL family to employ for the specific use case. Thus, the feasibility

and utility of the framework were demonstrated. This indicated that the primary objective of this study was sufficiently met.

9.3 Reflections on the proposed framework

This section will reflect on the proposed framework and its shortcomings.

In order to compare the NoSQL families, four very popular database products, one from each of the families, were used to represent the performance characteristics of the families. The researcher generalised the information of specific products to the NoSQL families, because non-generalised information may not exist. For example, the document-based family was represented by the *MongoDB* product. Therefore, the information used to compare the families can be seen as a shortcoming of the study. However, IT practitioners would also have to use generalised information to represent the families.

Since this is a Master's study, time was a restriction. The grading process especially was very time-consuming. The time spent grading the NoSQL families placed a limitation on this study. With more time, a more comprehensive evaluation could have been done, which may have led to a better recommendation. However, IT practitioners in the industry will not be able to spend large amounts of time finding and evaluating all relevant information either.

As a result of time constraints, only one instance was used to demonstrate the feasibility and utility of the framework. With more time, additional scenarios could have been used to demonstrate the effect of the framework on other specific instances.

One of the more difficult tasks of the proposed framework was assigning performance grades to the families according to the criteria. The grades that could be assigned ranged from 1 to 10. However, when grading was completed, no family received a grade close to either 1 or 10. Thus, it could be argued that the grading method used a smaller scale (4 to 9) to indicate the performance values of the NoSQL families.

The weighting of criteria was another difficult process. Many of the weights that were provided by the experts were close in value. Therefore, the majority of the weights did not contradict one another. This indicates that assigning weight values is not an easy task for experts to perform. The weight values may assist the IT practitioner in overcoming some of the decision-making biases. However, they may also support other biases. For example, a false sense of accuracy may be created when a decision is based on a calculation involving the weight values.

The conceptual data structure criterion captures two aspects pertaining to the general use of the data. The first is how the data looks after conversion, and the second is how the data can be used in general.

However, the criterion may not place enough focus on the use of the data in specific contexts, which could also influence the recommendation of the framework. Therefore, some of the criteria may not be granular enough to assign an appropriate weight value to accommodate the actual use of the data.

The recommendation of the framework is not perfect. In the NetFlow use case, the four families received relatively close final scores. The scores of two families were very close to each other, indicating these families could provide comparable levels of performance within the use case. While the recommendation from the framework may reduce uncertainty, it does not remove uncertainty completely. As the weights and gradings came from a smaller range than anticipated, uncertainty may not have been reduced enough for the recommendation to be trusted fully. Thus, the recommendation could result in a deeper investigation.

9.4 Future research

Future studies could investigate the rest of the design science framework of March and Smith (1995) within the context of this framework. This study only focused on the *build* research activity. However there are three other activities, namely *evaluate, theorise,* and *justify*, that could also be investigated. To evaluate means to determine if any progress has been made towards the goal of the artefact (March & Smith, 1995). To theorise means to develop theories regarding the framework. After the theories are created, they also need to be justified (March & Smith, 1995). As a result of their goals, theorise and justify do not provide value in the context of this study.

In terms of the NoSQL technology, this study had a focus on the four NoSQL families and the choice between them. However, a future research project could take the framework a step further and assist in the selection of a specific product within one of the families. Once a family is chosen, the next choice is to select a product since there are many products deriving from each family.

In terms of evaluation, a future research project could be performing a more comprehensive evaluation of the assignment of performance grades. Currently, each family's performance grades were assigned based on literature and argumentation. However, a more refined grade value could be provided if actual performance tests were done. The use case could also be changed to evaluate the effectiveness of the framework within other contexts.

Another future research project could be investigating the weighting process and the meaning of the weight values in more granular detail. This could assist IT practitioners in assigning weight values that indicate the importance of the criteria more easily. The fixed set of criteria should also be investigated further to account for various situations, as this could increase the quality of the recommendation.

Creating an application that will enable the framework to be used in industry without time constraints could also be a future research project. The application would aim to automate the framework to provide quick recommendations to the IT practitioners. The application would have a specific technology focus, namely the NoSQL focus found within this study. Therefore, the application would represent the framework used to assist IT practitioners with decisions regarding the NoSQL families. The goal of the research would be to measure the effectiveness of the framework in industry and determine whether IT practitioners would employ it. To enable such an application to be created, a more in-depth evaluation of the NoSQL families would have to be performed to determine a standardised set of performance figures for each family.

Since there are copious amounts of information about the NoSQL families and products available, another future research project could be creating a more consolidated piece of information pertaining to the NoSQL families, their levels of performance, their limitations and drawbacks, and common and uncommon use cases. The goal of such a research project would be to measure the effectiveness of this information in helping IT practitioners make better decisions regarding NoSQL technologies.

9.5 Final words

It was determined that decision-making is a difficult task to accomplish. Making decisions regarding technologies, such as NoSQL databases, is even more difficult, as there are large volumes of information to consider. Furthermore, biases and measurements can influence the IT practitioner and lead to incorrect decisions being made. Therefore, the problem statement of this study was: *IT practitioners do not have a systematic way to select a NoSQL family for non-arbitrary use cases*.

This study aimed to develop an artefact that can assist IT practitioners with decisions regarding the NoSQL families. To accomplish this goal, it proposed a framework that can mitigate the effects of biases and measurements while assisting IT practitioners in making more informed technology decisions.

This chapter reported the success of this research study. Each of the research sub-objectives was aligned with a part of the study, meaning that each sub-objective was met successfully. The primary research objective was also met by developing a framework that can assist IT practitioners in making better decisions regarding NoSQL technologies and thereby decrease the difficulty of making certain technology-based decisions.

BIBLIOGRAPHY

Abadi, D. J. (2009). Data management in the cloud: Limitations and opportunities. *IEEE Data Engineering Bulletin*, *32*(1), 3-12.

Abramova, V., & Bernardino, J. (2013). NoSQL databases: MongoDB vs Cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering* (pp. 14-22). ACM.

Abramova, V., Bernardino, J., & Furtado, P. (2014). Experimental evaluation of NoSQL databases. *International Journal of Database Management Systems*, *6*(3), 1-16.

Abubakar, Y., Adeyi, T. S., & Auta, I. G. (2014). Performance evaluation of NoSQL systems using YCSB in a resource austere environment. *International Journal of Applied Information Systems*, *7*(8), 23-27.

Agrawal, D., El Abbadi, A., Das, S., & Elmore, A. J. (2011). Database scalability, elasticity, and autonomy in the cloud. In *International Conference on Database Systems for Advanced Applications* (pp. 2-15). Springer Berlin Heidelberg.

Aiyer, A. S., Bautin, M., Chen, G. J., Damania, P., Khemani, P., Muthukkaruppan, K., ... & Vaidya, M. (2012). Storage infrastructure behind Facebook messages: Using HBase at scale. *IEEE Data Engineering Bulletin*, *35*(2), 4-13.

Aniceto, R., Xavier, R., Guimarães, V., Hondo, F., Holanda, M., Walter, M. E., & Lifschitz, S. (2015). Evaluating the Cassandra NoSQL database approach for genomic data persistency. *International Journal of Genomics*, *2015*. http://doi.org/10.1155/2015/502795

Audette, J. (2011). Overview - Scaling Out vs Scaling Up. Retrieved from https://www.mojoportal.com/overview-scaling-out-vs-scaling-up

Banker, K. (2011). *MongoDB in action*. Shelter Island, NY. Manning Publications Company.

Batra, S., & Tyagi, C. (2012). Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)*, *2*(2), 509-512.

Bazerman, M.H. and Moore, D.A. (2008), *Judgment in Managerial Decision Making* (7th ed.). New York: Wiley.

Benson, B. (2016). Cognitive bias cheat sheet – Better Humans. Retrieved from https://betterhumans.coach.me/cognitive-bias-cheat-sheet-55a472476b18

Brenner, L. A., Koehler, D. J., & Tversky, A. (1996). On the evaluation of one-sided evidence. *Journal of Behavioral Decision Making*, *9*(1), 59-70.

Brewer, E. A. (2000). Towards robust distributed systems. In *Symposium on Principles of Distributed Computing (PODC), 7*.

Brewer, E. A. (2012). CAP twelve years later: How the rules have changed. *IEEE Computer*, *45*(2), 23-29.

Cai, L., Huang, S., Chen, L., & Zheng, Y. (2013). Performance analysis and testing of HBase based on its architecture. In *12th International Conference on Computer and Information Science (ICIS), 2013 IEEE/ACIS* (pp. 353-358). IEEE.

Carlson, J. L. (2013). *Redis in Action*. Shelter Island, NY: Manning Publications Company.

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM Special Interest Group on Management of Data (SIGMOD) Record*, *39*(4), 12-27.

Chapple, M. (2018). The Basics of Normalizing a Database. Retrieved from https://www.thoughtco.com/database-normalization-basics-1019735

Chen, M., Mao, S., & Liu, Y. (2014). Big data: a survey. *Mobile Networks and Applications*, *19*(2), 171-209.

Chodorow, K. (2013). *MongoDB: the definitive guide*. Sebastopol, CA: O'Reilly Media, Incorporated.

Churchill Jr, G. A. (1979). A paradigm for developing better measures of marketing constructs. *Journal of Marketing Research (JMR)*, 64-73.

Cisco. (2012). Introduction to Cisco IOS NetFlow - A Technical Overview. Retrieved from http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

Clemen, R. T., & Gregory, R. (1995). Creative decision making: A handbook for active decision makers. *Eugene, OR: Decision Science Research Institute*.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (pp. 143-154). ACM.

Coronel, C., & Morris, S. (2016). *Database systems: Design, implementation, and management* (12th edition [student edition].). Stamford, Conn.: Cengage Learning.

Couchbase. (n.d.). Comparing document-oriented and relational data. Retrieved from https://developer.couchbase.com/documentation/server/3.x/developer/dev-guide-3.0/compare-docs-vs-relational.html

Cowan, R. (1991). Tortoises and hares: choice among technologies of unknown merit. *The Economic Journal*, *101*(407), 801-814.

Da Silva, M. D., & Tavares, H. L. (2015). *Redis Essentials*. Birmingham, UK: Packt Publishing Ltd.

Das, V. (2015). *Learning Redis*. Birmingham, UK: Packt Publishing Ltd.

DB-Engines. (n.d.). DB-Engines Ranking. Retrieved from https://db-engines.com/en/ranking

Dean, R. B., & Dixon, W. J. (1951). Simplified statistics for small numbers of observations. *Analytical Chemistry*, *23*(4), 636-638.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., ... & Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *ACM Special Interest Group on Operating Systems (SIGOPS) Review*, *41*(6), 205-220.

Desouza, K., Jha, S., Papagari, S., & Ye, C. (2006). Choosing between technology solutions. *Engineering Management Journal*, *16*(1), 42-45.

Dimiduk, N., Khurana, A., Ryan, M. H., & Stack, M. (2013). *HBase in action*. Shelter Island, NY: Manning Publications Company.

Dobelli, R. (2013). *The art of thinking clearly: better thinking, better decisions*. Hachette, UK: Sceptre Publishers.

Domaschka, J., Hauser, C. B., & Erb, B. (2014). Reliability and availability properties of distributed database systems. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC)*, (pp. 226-233). IEEE.

Du Toit, P. (2016). An evaluation of non-relational database management systems as suitable storage for user generated text-based content in a distributed environment, University of South Africa, Pretoria, http://hdl.handle.net/10500/21613

Edlich, S. (2011). List of NoSQL databases [currently >225]. Retrieved from http://nosql-database.org/

Evans, E. (2009, May 12). NoSQL 2009 - Blog-post of 2009-05-12. Retrieved January 13, 2017, from http://blog.sym-link.com/2009/05/12/nosql_2009.html

Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, *35*(2), 137-144.

Gao, X., Nachankar, V., & Qiu, J. (2011). Experimenting lucene index on HBase in an HPC environment. In *Proceedings of the First Annual Workshop on High Performance Computing Meets Databases* (pp. 25-28). ACM.

George, L. (2011). *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. Sebastopol, CA: O'Reilly Media, Incorporated.

Gilbert, S., & Lynch, N. A. (2012). Perspectives on the Cap theorem. *Computer*, *45*(2), (pp. 30-36).

Gillham, B. (2011). *Developing a questionnaire* (2nd ed.). New York, NY: Continuum International Publishing Group.

Goel, A. (2015). *Neo4j Cookbook: harness the power of Neo4j to perform complex data analysis over the course of 75 easy-to-follow recipes*. Birmingham, UK: Packt Publishing Ltd.

Gold, J. I., & Shadlen, M. N. (2007). The neural basis of decision making. *Annual Review of Neuroscience*, *30*, 535-574.

Goldman, A. E., & McDonald, S. S. (1987). *The group depth interview: Principles and practice*. Englewood Cliffs, NJ: Prentice-Hall.

Gravetter, F. J., & Wallnau, L. B. (2011). *Essentials of Statistics for the Behavioral Sciences* (7th ed.). Belmont, CA: Wadsworth Cengage.

Gu, Y., Wang, X., Shen, S., Ji, S., & Wang, J. (2015). Analysis of data replication mechanism in NoSQL database MongoDB. In *2015 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, (pp. 66-67). IEEE.

Guo, H., Wang, L., Chen, F., & Liang, D. (2014). Scientific big data and digital earth. *Chinese Science Bulletin, 59*(35), 5066-5073.

Győrödi, C., Győrödi, R., & Sotoc, R. (2015a). A comparative study of relational and non-relational database models in a web- based application. *International Journal of Advanced Computer Science and Applications (IJACSA), 6*(11), 78–83. http://doi.org/10.14569/IJACSA.2015.061111

Győrödi, C., Győrödi, R., Pecherle, G., & Olah, A. (2015b). A comparative study: MongoDB vs. MySQL. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES),* (pp. 1-6). IEEE.

Hahn, U., & Harris, A. J. L. (2014). *What does it mean to be biased: motivated reasoning and rationality. Psychology of Learning and Motivation - Advances in Research and Theory* (1st ed., Vol. 61). Elsevier Incorporated. http://doi.org/10.1016/B978-0-12-800283-4.00002-2

Hair, J. F., Black, W. C., Babin, B. J. & Anderson, R. E. (2010). Multivariate data analysis: A global perspective. (7th ed.). Upper Saddle River, NJ: Pearson Education.

Hammond, J. S., Keeney, R. L., & Raiffa, H. (1998). The hidden traps in decision making. *Harvard Business Review, 76*(5), 47-58.

Han, J., Cai, Y., & Cercone, N. (1993). Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering, 5*(1), 29-40.

Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. In *2011 6th International Conference on Pervasive computing and applications (ICPCA),* (pp. 363-366). IEEE.

HBase. (2007). Apache HBase – Apache HBase™. Retrieved from https://hbase.apache.org/

Hecht, R., & Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing (CSC).* (pp. 336-341). IEEE.

Hoff, T. (2011, June 20). 35 Use cases for choosing your next NoSQL database - high scalability -. Retrieved February 9, 2017, from http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html

Hu, H., Wen, Y., Chua, T. S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, *2*, 652–687. http://doi.org/10.1109/ACCESS.2014.2332453

Hubbard, D. W. (2011). *How to measure anything workbook: finding the value of intangibles in business*. Hoboken, NJ: John Wiley & Sons.

Hwang, J. S., Lee, S., Lee, Y., & Park, S. (2015). A selection method of database system in bigdata environment: a case study from smart education service in Korea. *International Journal of Advances in Soft Computing and its Applications (IJASCA)*, *7*(1), 9-21.

Iravani, M. (2015). How Twitter Uses Redis To Scale - 105TB RAM, 39MM QPS, 10,000 Instances s. Retrieved from https://www.linkedin.com/pulse/how-twitter-uses-redis-scale-105tb-ram-39mm-qps-10000-iravani

ITBusinessEdge. (n.d.). Top five NoSQL databases and when to use them. Retrieved from http://www.itbusinessedge.com/slideshows/top-five-nosql-databases-and-when-to-use-them-07.html

Jacobson, I. (2003). Use cases and aspects-working seamlessly together. *Journal of Object Technology*, *2*(4), 7-28.

Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D. (2012). A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, *1*(6).

Jiang, Y. (2012). *HBase Administration Cookbook: master HBase configuration and administration for optimum database performance*. Birmingham, UK: Packt Publishing.

Jouili, S., & Vansteenberghe, V. (2013). An empirical comparison of graph databases. In *2013 International Conference on Social Computing (SocialCom)*, (pp. 708-715). IEEE.

Kahneman, D. (2000). A psychological point of view: Violations of rational rules as a diagnostic of mental processes. *Behavioral and Brain Sciences*, *23*(5), 681-683.

Kahneman, D., & Tversky, A. (1979). Prospect Theory: an analysis of decision under risk. *Econometrica*, *47*(2), 263–292. https://doi.org/10.2307/1914185

Kahneman, D., & Tversky, A. (1984). Choices, values, and frames. *American Psychologist*, *39*(4), 341-350.

Kahneman, D., Knetsch, J. L., & Thaler, R. H. (1991). Anomalies: The endowment effect, loss aversion, and status quo bias. *The Journal of Economic Perspectives*, *5*(1), 193-206.

Kahneman, D., Wakker, P. P., & Sarin, R. (1997). Back to Bentham? Explorations of experienced utility. *The Quarterly Journal of Economics*, *112*(2), 375-406.

Karger, D., Sherman, A., Berkheimer, A., Bogstad, B., Dhanidina, R., Iwamoto, K., ... & Yerushalmi, Y. (1999). Web caching with consistent hashing. *Computer Networks*, *31*(11), 1203-1213.

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

Kemper, C. (2015). *Beginning Neo4j*. Berkeley, CA: Apress.

Khetrapal, A., & Ganesh, V. (2006). HBase and Hypertable for large scale distributed storage systems. *Department of Computer Science, Purdue University*, 21-28.

Kuhlenkamp, J., Klems, M., & Röss, O. (2014). Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment*, *7*(12), 1219-1230.

Kulak, D., & Guiney, E. (2012). *Use cases: requirements in context*. Boston, MA: Addison-Wesley.

Kvale, S. (2008). *Doing interviews*. Thousand Oaks, CA: Sage.

Lakkaraju, K., Yurcik, W., & Lee, A. J. (2004). NVisionIP. *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security - VizSEC/DMSEC*, 65. http://doi.org/10.1145/1029208.1029219

Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer*, *43*(2), 12-14.

LeBoeuf, R. A., & Shafir, E. (2006). The long and short of it: Physical anchoring effects. *Journal of Behavioral Decision Making*, *19*(4), 393-406.

Li, Y., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, (pp. 15-19). IEEE.

Lourenço, J. R., Abramova, V., Vieira, M., Cabral, B., & Bernardino, J. (2015a). Nosql databases: A software engineering perspective. In *3rd World Conference on Information Systems and Technologies (WorldCIST'15)*, (pp. 741-750). Springer International Publishing.

Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., & Bernardino, J. (2015b). Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data*, *2*(1), 18. https://doi.org/10.1186/s40537-015-0025-0

Lumsdaine, E., & Lumsdaine, M. (1994). Creative problem solving. *IEEE Potentials*, *13*(5), 4-9.

Magee, J. F. (1964). *Decision trees for decision making* (pp. 35-48). Harvard Business Review.

Magnusson, H. (2013, October 11). Common MongoDB Use Cases. Retrieved from https://www.slideshare.net/mongodb/common-use-cases-hannes

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251-266.

Mayo, M. (2016, June). Top NoSQL database engines. Retrieved from http://www.kdnuggets.com/2016/06/top-nosql-database-engines.html

Medjahed, B., Ouzzani, M., & Elmagarmid, A. K. (2009). Generalization of ACID Properties. In *Encyclopedia of Database Systems* (pp. 1221-1222). Springer US.

Merriam-Webster. (n.d.). Bias. Retrieved from https://www.merriam-webster.com/dictionary/bias

Microsoft. (2005). Understanding availability, reliability, and scalability. Retrieved from https://technet.microsoft.com/en-us/library/aa996704(v=exchg.65).aspx

Minarik, P., & Dymacek, T. (2008). NetFlow Data Visualization Based on Graphs. In J. R. Goodall, G. Conti, & K.-L. Ma (Eds.), *Visualization for Computer Security: 5th International Workshop, VizSec 2008, Cambridge, MA, USA, September 15, 2008. Proceedings* (pp. 144–151). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85933-8_14

MongoDB. (2008). The MongoDB 3.4 Manual. Retrieved from https://docs.mongodb.com/manual/

Moniruzzaman, A.B.M. & Hossain, S.A. (2013). NoSQL database: new era of databases for big data analytics – classification, characteristics and comparison, *International Journal of Database Theory and Application*, *6*(4), pp.1–14.

Montag, D. (2013). Understanding Neo4j scalability. Retrieved from http://info.neo4j.com/rs/neotechnology/images/Understanding%20Neo4j%20Scalability%282%29.pdf?_ga=1.160766525.1090831026.1491384086

Morewedge, C. K., & Kahneman, D. (2010). Associative processes in intuitive judgment. *Trends in Cognitive Sciences*, *14*(10), 435-440.

Morgan, D. (1996). Focus groups. *Annual Review of Sociology*,*22*, 129-152. Retrieved from http://www.jstor.org/stable/2083427

Naheman, W., & Wei, J. (2013). Review of NoSQL databases and performance testing on HBase. In *2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, (pp. 2304-2309). IEEE.

Nayak, A. (2014). *MongoDB cookbook: over 80 practical recipes to design, deploy, and administer MongoDB*. Birmingham, UK: Packt Publishing.

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

Nelubin, D., & Engber, B. (2013). NoSQL failover characteristics: Aerospike, cassandra, couchbase, mongodb. *Thumbtack Technology*, 1-19.

Neo4j. (2017). The Neo4j Operations Manual v3.3. Retrieved from https://neo4j.com/docs/operations-manual/3.3/

Newton, P. (2016). 6 Key Decision Making Techniques. Retrieved from http://www.free-management-ebooks.com/dldebk-pdf/fme-6-decision-making-techniques.pdf

Nishimura, S., Das, S., Agrawal, D., & El Abbadi, A. (2011). MD-HBase: A scalable multi-dimensional data infrastructure for location aware services. In *2011 12th IEEE International Conference on Mobile Data Management (MDM), 1*, pp. 7-16. IEEE.

Nooraie, M. (2012). Factors influencing strategic decision-making processes. *International Journal of Academic Research in Business and Social Sciences*, *2*(7), 405-429.

Nunnally, J. C. (1967). *Psychometric theory*. New York, USA: McGraw-Hill.

Nunnally, J. C., & Bernstein, I. H. (1994). *Psychometric theory* (3rd ed.). New York, USA: McGraw-Hill.

Olivier, M. S. (2009). *Information technology research – a practical guide for computer science and informatics* (3rd ed.). Pretoria, SA: Van Schaik Publishers.

O'Reilly, C. A. (1980). Individuals and information overload in organizations: is more necessarily better?. *Academy of Management Journal*, *23*(4), 684-696.

Orend, K. (2010). *Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer.* (Master Thesis), Technical University of Munich, Munich. Retrieved from https://weblogs.in.tum.de/file/ikcuitkq8cpm/Publications/2010/Or10/Or10.pdf

Oxford English Dictionary. (2017a). Bias - definition of bias in English | Oxford Dictionaries. Retrieved from https://en.oxforddictionaries.com/definition/bias

Oxford English Dictionary. (2017b). Measure - definition of measure in English | Oxford Dictionaries. Retrieved from https://en.oxforddictionaries.com/definition/measure

Oxford English Dictionary. (2017c). Decision | Definition of decision in English by Oxford Dictionaries. Retrieved from https://en.oxforddictionaries.com/definition/decision

Padhy, R. P., Patra, M. R., & Satapathy, S. C. (2011). RDBMS to NoSQL: Reviewing some next-generation non-relational databases. *International Journal of Advanced Engineering Sciences and Technologies (IJAEST)*, *11*(1), 15-30.

Piplani, A. (2010, May 6). U pick 2 selection for NoSQL providers. Retrieved November 20, 2016, from http://amitpiplani.blogspot.co.za/2010/05/u-pick-2-selection-for-nosql-providers.html

Plugge, E., Hows, D., Membrey, P., & Hawkins, T. (2015). *The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB*. California: Apress.

Pokorny, J. (2013). NoSQL databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, *9*(1), 69-82.

Qu, S. Q., & Dumay, J. (2011). The qualitative research interview. *Qualitative Research in Accounting & Management*, *8*(3), 238-264.

Redis. (2017). Redis Documentation. Retrieved from https://redis.io/documentation

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: new opportunities for connected data* (2nd ed.). Sebastopol, CA: O'Reilly Media, Incorporated.

Rokach, L., & Maimon, O. (2005). Decision trees. In *Data mining and knowledge discovery handbook* (pp. 165-192). Springer, Boston, MA.

Russo, J. E., Schoemaker, P. J., & Russo, E. J. (1989). *Decision traps: Ten barriers to brilliant decision-making and how to overcome them*. New York, NY: Doubleday/Currency Publishers.

Sasaki, B. M. (2015, September 11). Graph databases for beginners: a tour of aggregate stores. Retrieved November 23, 2016, from https://neo4j.com/blog/aggregate-stores-tour/

SASO. (2006). Guide to the expression of uncertainty in measurement. Retrieved from http://chapon.arnaud.free.fr/documents/resources/stat/GUM.pdf

Scheubrein, R., & Zionts, S. (2006). A problem structuring front end for a multiple criteria decision support system. *Computers & Operations Research*, *33*(1), 18-31.

Schmidt, R. D. O., Sperotto, A., Sadre, R., & Pras, A. (2012). Towards bandwidth estimation using flow-level measurements. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7279*, 127–138. https://doi.org/10.1007/978-3-642-30633-4_18

Seguin, K. (2012). *The Little Redis Book*. Retrieved from http://openmymind.net/redis.pdf

Sharma, V., & Dave, M. (2012). SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(8), 20-27.

Shon, P. (2014). Redis Explained in 5 Minutes or Less. Retrieved from https://www.credera.com/blog/technology-insights/java/redis-explained-5-minutes-less/

Sommer, R., & Feldmann, A. (2002). NetFlow: Information loss or win? In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, 173–174. https://doi.org/10.1145/637201.637226

Speier, C., Valacich, J. S., & Vessey, I. (1999). The influence of task interruption on individual decision making: An information overload perspective. *Decision Sciences*, *30*(2), 337-360.

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP flow-based intrusion detection. *IEEE Communications Surveys & Tutorials*, *12*(3), 343-356.

Stephens, R., Plew, R., & Jones, A. D. (2009). *Sams teach yourself SQL in one hour a day*. Sams Publishing.

Stewart, D. W., & Shamdasani, P. N. (2014). *Focus groups: Theory and practice* (Vol. 20). London, UK: Sage publications.

Stonebraker, M. (1986). The case for shared nothing. *IEEE Data Engineering Bulletin*, *9*(1), 4–9.

Strack, F., Martin, L. L., & Schwarz, N. (1988). Priming and communication: Social determinants of information use in judgments of life satisfaction. *European Journal of Social Psychology (EJSP)*, *18*(5), 429-442.

Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*.

Strozzi, C. (2010). NoSQL - A Relational Database Management System. Retrieved from http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

Tauro, C. J., Aravindh, S., & Shreeharsha, A. B. (2012). Comparative study of the new generation, agile, scalable, high performance NOSQL databases. *International Journal of Computer Applications*, *48*(20), 1-4.

Thaler, R. (1980). Toward a positive theory of consumer choice. *Journal of Economic Behavior & Organization*, *1*(1), 39-60.

Tutorialspoint. (n.d.). Redis Partitioning. Retrieved from https://www.tutorialspoint.com/redis/redis_partitioning.htm

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, *185*(4157), 1124–1131.

Tversky, A., & Kahneman, D. (1985). The framing of decisions and the psychology of choice. In *Environmental Impact Assessment, Technology Assessment, and Risk Analysis*, 107-129. Springer Berlin Heidelberg.

Vogels, W. (2009). Eventually consistent. *Communications of the ACM* (52)1, 40-44. https://doi.org/10.1145/1435417.1435432

Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., & Partner, J. (2015). *Neo4j in action*. Shelter Island, NY: Manning Publications Company.

Wellhausen, T. (2012). Highly scalable, ultra-fast and lots of choices a pattern approach to NoSQL. Retrieved from http://www.tim-wellhausen.de/papers/NoSQL-Patterns/NoSQL-Patterns.html

Yu, S. (2009). Acid properties in distributed databases. *Advanced eBusiness Transactions for B2B-Collaborations*.

Zhang, Q. (2011). Research based on cloud database, *Management Expert*. *12*(15).

Zhou, X., Petrovic, M., Eskridge, T. & Carvhalo, M. (2014). Exploring Netflow data using Hadoop, In *Proceedings of the Second ASE International Conference on Big Data Science and Computing*, Academy of Science and Engineering (ASE), US, 2014. CA: Stanford. 1-10.

APPENDIX A: RESEARCH REGARDING BIASES IN DECISION-MAKING

Biases	Sources	Applicability to IT
Status Quo	[1] [8] [13] [14]	*****
Anchoring	[1] [5] [6] [14]	*****
Sunk-Cost	[1] [8] [14] [16]	*****
Confirming Evidence	[1] [4] [10] [11] [14]	*****
Framing	[1] [3] [12] [14]	*****
Estimating and Forecasting	[1] [14] [15]	**
Overconfidence	[1] [4] [14]	****
Prudence	[1] [14]	*****
Prediction	[2] [6] [15]	**
Recallability	[1] [9] [14]	*****
Plunging in	[7]	****
Frame blindness	[7] [12]	*
Lack of frame control	[7] [10] [14]	***
Overconfidence in our judgement	[7] [14]	****
Shortsighted shortcuts	[7]	***
Shooting from the hip	[7]	*****
Group failure	[7] [14]	**
Fooling ourselves about feedback	[7]	***
Not keeping track	[7] [14]	****
Failure to audit decision process	[7]	*****
Reciprocity	[14]	*
Halo Effect	[14]	*****
Expectations	[14]	***

Number	Citation
[1]	Hammond, J. S., Keeney, R. L., & Raiffa, H. (1998). The hidden traps in decision making. *Harvard business review*, *76*(5), 47-58.
[2]	Kahneman, D., & Tversky, A. (1973). On the psychology of prediction. *Psychological review*, *80*(4), 237.
[3]	Tversky, A., & Kahneman, D. (1985). The framing of decisions and the psychology of choice. In *Environmental Impact assessment, technology assessment, and risk analysis* (pp. 107-129). Springer, Berlin, Heidelberg.
[4]	Kahneman, D., & Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the econometric society*, 263-291.
[5]	LeBoeuf, R. A., & Shafir, E. (2006). The long and short of it: Physical anchoring effects. *Journal of Behavioral Decision Making*, *19*(4), 393-406.
[6]	Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. Science, 185, 1124–1131.
[7]	Russo, J. E., Schoemaker, P. J., & Russo, E. J. (1989). *Decision traps: Ten barriers to brilliant decision-making and how to overcome them*. New York, NY.: Doubleday/Currency.

[8]	Kahneman, D., Knetsch, J. L., & Thaler, R. H. (1991). Anomalies: The endowment effect, loss aversion, and status quo bias. *The journal of economic perspectives*, *5*(1), 193-206.
[9]	Kahneman, D., Wakker, P. P., & Sarin, R. (1997). Back to Bentham? Explorations of experienced utility. *The quarterly journal of economics*, *112*(2), 375-406.
[10]	Morewedge, C. K., & Kahneman, D. (2010). Associative processes in intuitive judgment. *Trends in cognitive sciences*, *14*(10), 435-440.
[11]	Brenner, L. A., Koehler, D. J., & Tversky, A. (1996). On the evaluation of one-sided evidence. *Journal of Behavioral Decision Making*, *9*(1), 59-70.
[12]	Strack, F., Martin, L. L., & Schwarz, N. (1988). Priming and communication: Social determinants of information use in judgments of life satisfaction. *European journal of social psychology*, *18*(5), 429-442.
[13]	Bazerman, M.H. and Moore, D.A. (2008), Judgment in Managerial Decision Making, 7th ed. New York: Wiley.
[14]	Dobelli, R. (2013). *The art of thinking clearly: better thinking, better decisions*. Hachette UK.
[15]	Tversky, A., & Kahneman, D. (1971). Belief in the law of small numbers. *Psychological bulletin*, *76*(2), 105.
[16]	Thaler, R. (1980). Toward a positive theory of consumer choice. *Journal of Economic Behavior & Organization*, *1*(1), 39-60.

Definition of each bias:

The *anchoring* trap refers to a situation where a business decision is influenced by information from past trends or events. The bulk of the decision is influenced by past events which mean that other factors are ignored. If an individual were asked to provide the population amount of South Africa, the individual might not be able to provide an exact number. However, the individual may recall information they read which had a number of the population and then use this recollection as the answer. The answer might be close to the correct amount, but in most cases, a factual number will not be provided. Thus, an anchor is used to base decisions on and affect the answer provided.

The *status-quo* trap refers to the biases that individuals have regarding the current state of affairs. Individuals would stick with the status quo and have a bias against the alternatives since the alternative involve too much risk. An individual is comfortable with a specific technology; however, work requires the individual to learn and use another technology. The individual knows nothing about the new technology and feels he is good enough with the current technology to not have to learn the new technology. Thus, a biased opinion is formed against the new technology even if it is the perfect solution.

The *sunk-cost* trap is where business decisions are made to justify past business decisions, even when these past decisions are not valid anymore. An example is finances spent on technology, that is not relevant anymore, but management refused to change the technology because of the cost. An individual invests 1000 rands in shares. The investment fails since it is now worth 750 rand.

However, the individual does not sell the shares and hopes the price of the shares will increase again. The individual could sell the shares and invest the money to make more than 1000 off other profitable shares.

The *confirming-evidence* trap is where individuals are looking for information that endorses and supports their knowledge. The individual assumes the acquired information is correct while not considering the alternative. The individual is looking for a reason to accept the confirming information and not question/consider opposing information. A board meeting will rarely discuss facts and information that oppose the views of the board. The focus would be on evidence or facts that confirm the board's views.

The *framing* of a question or problem can influence the way a business decision is made. If the emphasis of the problem or question is incorrect, the solution might be incorrect too. The *framing* trap can influence the success or failure of a project. Individuals interpret the problem incorrect since the emphasis is incorrectly placed in the problem or question.

The *estimating and forecasting* trap is where forecasting and estimating is done without the factual information. *Estimating and forecasting* can lead to incorrect business decisions or mistakes being made. An example of *estimating and forecasting* trap is with estimating the prices of fuel in 50 years. There is not sufficient information to make an accurate forecast for such a scenario.

Overconfidence can lead to a bias decision since other possibilities are not taken into account. *Overconfidence* means that a proper solution to a problem might not be looked at since an individual is overconfident about their predictions. An individual that is *overconfident* in their abilities and knowledge may not consider the alternatives to decision-making situations. A proper solution to a problem can be overlooked because of *overconfidence*.

The *prudence* trap is where individuals are over cautious when high-risk decisions need to be made. Individuals can lead a project to failure if the proper decisions are not made since the individual wants to be safe. A high-risk decision may have many rewards if the correct choices are made. The high-risk decision may also have many penalties if the incorrect choices are made. An example of this trap is in weapons design. Engineers design rugged weapons to handle the worst conditions, however, the weapons hardly ever operate in harsh conditions.

The *recallability* trap is when past events or dramatic occurrences in an individual's life influences the business decisions the individual must make. This influence can lead to biased decisions which influence the success of a project or business decision. An example of this is when individuals see a plane crash on television. The memory of the accident will influence the individual's decision about

whether to take a flight or take a boat to reach their destination. Past events are recalled to base decisions on, and this can lead to incorrect decisions being made.

Plunging in is where conclusions are drawn prematurely without sufficient knowledge about the problem and how to solve the problem. Individuals do not stop and think about the crux of the issue and which information to collect before making premature conclusions. Individuals work on a crucial issue of a project. The individual does not step back and consider the actual problem which can lead to wrong decisions being made. The individual should ask secondary questions, for example; what is the primary problem, how much time do I estimate till completion, and how do I think such a decision should be made.

Frame blindness is where individuals' base decisions on their mental framework. The mental framework considers all information regarding the problem and refers to how the individual thinks about the decision(s) to be made; what must be decided, what are the options, and what are the criteria for choosing between options. Their mental framework could be wrong since not enough information is present which could lead to overlooking the correct decisions. The *framing* of the problem has a large influence on this decision-making mistake.

Lack of frame control occurs when individuals influence the decision-making process of others. The influence of others can affect the decisions made by introducing biases to the decision-making process. *Lack of frame control* can also occur if an individual cannot consciously define the problem in different ways. The individual does not understand the problem and adopts other individual's mental frameworks for decision-making.

Overconfidence in our judgement is when individuals are overconfident in their knowledge without collecting the necessary information. Decisions made with overconfidence can result in an incorrect decision and wrong solutions being chosen. An overconfident individual value their knowledge more than the factual information and may not consider the alternative. The decisions are based on their knowledge which can be incorrect.

Shortsighted shortcuts are a barrier to decision-making since anchors influence the decision being made. The most readily available information is trusted to be correct while other opposing information is not considered. This decision-making mistake can result in biased decision-making since the alternative is not investigated.

Shooting from the hip is a barrier where individuals do not follow a systematic decision process to solve a problem. An individual is presented with a problem for a project. The individual's mind processes all of the possessed information quickly, without properly noting the factual information.

A conclusion is drawn based on all the processed information and accepted as correct. However, important facts and information may not be noted/investigated which leads to incorrect decisions being made. Individuals not following a systematic decision process have the risk of making the incorrect decisions.

Group failure is a barrier that occurs in decisions being made by groups of individuals. The assumption is that groups of individuals will make the correct decision. If members of a group agree prematurely on an answer, it may be the incorrect answer if not all factual information is investigated. Some individuals within the group influence others to agree with their decision, although the decision may be incorrect. If there is no group-decision process being followed, the decision-making can be biased and incorrect. An example of the *group failure* barrier is when 50 million people agree that an answer is correct. However, it does not mean the answer is correct.

Fooling ourselves about feedback is when individuals do not consider past events or occurrences when making business decisions. Ignoring past feedback could lead to information being missed or left out that can influence the success of decisions. If an individual ignores past experiences with decision-making, the individual might make the same mistakes since no hindsight is gained about the problem. An individual purchases a car with black paint. The individual experiences extreme heat and has an unpleasant experience. The individual buys another car with black paint. Making the same decision to buy a black paint car, implies that the individual did not learn from past experiences and did not have hindsight regarding the problem.

Not keeping track of decisions and their outcomes is another barrier to decision-making. Past decisions and their outcomes can teach lessons about what to consider for future decision-making. If an individual makes a mistake in the past, the individual is not expected to make the same mistake twice. However, an individual does not keep track of past mistakes/decisions and makes the same mistake as in the past. Making the same mistake implies that the individual did not keep track of the outcomes of past decisions or experiences. The *not keeping track* barrier is closely related to *fooling ourselves about feedback* barrier to decision-making.

Failure to audit decision process is another barrier to decision-making. Not following a decision process can make a decision-maker vulnerable to numerous barriers and traps. A decision-process is there to assist an individual to understand the problem, the different choices to be made and how to implement the decisions. If an individual does not follow such a decision process, the wrong decisions may be made since the individual may not understand the problem.

Prediction can be a barrier to decision-making when individuals make predictions under uncertainty. Individuals make predictions based on many factors to predict an event that might happen. If not,

enough factual information is considered, the prediction is made under uncertainty which means it is more than likely to not be true. Individuals can make predictions about finances of a company, however, may be wrong if the incorrect or too little information is used.

Reciprocity refers to an exchange of services/goods for other services/goods. If an individual purchase a service from another individual, money is provided in exchange for a service being provided. The exchange of services or items is what *reciprocity* refers to. Reciprocity becomes a barrier to decision-making when an individual is in debt to another individual. Decisions made are influenced by feelings of guilt/debt and can lead to incorrect decisions being made.

Halo-effect means that one aspect can affect the way individuals see the whole picture. If one aspect produced a positive or negative feeling, the feeling could influence the individual's decision about a specific topic. The *Halo-effect* could lead to a biased decision that affects the success of the project. Depending on the feeling an individual receives, the choice about the topic may be influenced, and other alternatives may not be investigated.

Expectations can influence the way individuals make decisions. If an individual has expectations, it means that the individual is expecting a certain result/occurrence to happen. If the expectation(s) of the individual are not met, the individual may judge the topic with a negative feeling and could influence the decisions after that. However, if the expectations are met, the individual may be biased towards the topic and not consider other alternatives.

APPENDIX B: WEIGHTING CRITERIA INSTRUMENT

Dear Respondent,

My name is Leon Rheeder. I am currently working towards my Masters in Information Technology under the supervision of Professor Reinhardt Botha at Nelson Mandela University in Port Elizabeth. My dissertation is titled "A Framework for Selecting NoSQL Databases: A NetFlow Use Case".

My aim is to assist IT practitioners in making more informed decisions when choosing between NoSQL databases. In my dissertation, I propose a weighted decision-making model that can be used to help achieve this goal. Weights will be assigned to various criteria that have been created based on current theory. Allocating weights is an integral part of the model and I suggest using a data collection tool to do this.

Since I am using NetFlow as a use case to demonstrate the model, I would like to make use of your networking expertise. For the purposes of this exercise, please position yourself as a possible technical team member that must decide which NoSQL database to use to store captured NetFlow data for decision making purposes. Normal operational use of NetFlow data will continue. However, storing the data in a NoSQL database will allow it to be used for additional purposes, such as trend analysis, security analysis, and decision support.

The instrument consists of only one question that asks you to weigh 9 possible criteria for NoSQL databases. You are allocated 50 marks to distribute between the 9 categories. Any individual criterion can be awarded a maximum of 10 marks.

I believe that this exercise will take up no more than 15 minutes of your time. Of course, participation is voluntary and you may choose not to participate. If you have any questions, please contact me telephonically on +27812700982 or via email at rheedera@icloud.com.

I greatly appreciate your participation in this exercise.

Kind regards,
Leon Rheeder

Next

* **Please give each of the nine criteria a rating of 1 – 10 to indicate the relative importance of each criterion when using a NoSQL database to store and use NetFlow data for, for example, trend analysis, security analysis, or decision support.**

PLEASE NOTE: You have only 50 (FIFTY) marks to distribute. In other words, the total of all the ratings must equal 50.

Read performance When analysing NetFlow data, it may be important that the read queries be performed quickly. A high rating indicates that fast reading from the NoSQL database is important for the NetFlow Big Data analysis use case.	1 –Least important		10 –Most important
Write performance When storing NetFlow data, it may be important that the write function of the NoSQL database be fast and effective. A high rating indicates that fast and effective writing to the NoSQL database is important for the NetFlow Big Data analysis use case.			
Consistency When analysing NetFlow data, it may be important that the data in the NoSQL database be consistent with the most recent flow data. A high rating indicates that the NoSQL database must be able to reflect the updated data immediately.			
Availability When analysing NetFlow data, it may be important that the NoSQL database be able to operate uninterruptedly. A high rating indicates that the NoSQL database must be able to operate accurately at all times.			
Partitioning Working with large volumes of NetFlow data may require different nodes across the network to be clustered together. This will allow data to be stored locally first and combined later. A high rating indicates that the NoSQL database must be able to cope with the addition and/or removal of nodes within a cluster.			
Scalability When storing and analysing NetFlow data, it may be important that the NoSQL database be able to accommodate an increase in capacity needs. A high rating indicates that it is important that the NoSQL database be able to deal with a sudden increase in the volume of the NetFlow data and/or the number of analysis queries performed.			
Data structure The structure of data may require that the data be transformed before storing it. NetFlow data is semi-structured. A high rating indicates that it is important that the NoSQL database require as little data transformation as possible.			
Reliability When analysing NetFlow data, it may be important that the NoSQL database be failure resistant and able to consistently perform its function. A high rating indicates that the NoSQL database must be highly fault tolerant when analysing NetFlow Big Data.			
Learning curve The amount of time and effort needed to set up and use the NoSQL database may have an impact on the NoSQL database technology selection. A high rating indicates that the learning curve has a large impact on the selection of NoSQL database technologies.			

0

Values must add up to 50

Done