A METHODOLOGY FOR EVALUATING RELATIONAL AND NOSQL DATABASES FOR SMALL-SCALE STORAGE AND RETRIEVAL

DISSERTATION

Ryan D. L. Engle, Major, USAF
AFIT-ENV-DS-18-S-047

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-DS-18-S-047

The views expressed in this paper are those of the author and do not reflect official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT-ENV-DS-18-S-047

A METHODOLOGY FOR EVALUATING RELATIONAL AND NOSQL DATABASES FOR SMALL-SCALE STORAGE AND RETRIEVAL

DISSERTATION

Presented to the Faculty

Department of Systems and Engineering Management

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Ryan D. L. Engle, BS, MS
Major, USAF

September 2018

AFIT-ENV-DS-18-S-047

A METHODOLOGY FOR EVALUATING RELATIONAL AND NOSQL DATABASES FOR SMALL-SCALE STORAGE AND RETRIEVAL

Ryan D. L. Engle, BS, MS
Major, USAF

Committee Membership:

Brent T. Langhals, PhD
Chairman

Michael R. Grimaila, PhD, CISM, CISSP
Member

Douglas D. Hodson, PhD
Member

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

Abstract

Modern systems record large quantities of electronic data capturing time-ordered events, system state information, and behavior. Subsequent analysis enables historic and current system status reporting, supports fault investigations, and may provide insight for emerging system trends. Unfortunately, the management of log data requires ever more efficient and complex storage tools to access, manipulate, and retrieve these records. Truly effective solutions also require a well-planned architecture supporting the needs of multiple stakeholders. Historically, database requirements were well-served by relational data models, however modern, non-relational databases, i.e. NoSQL, solutions, initially intended for "big data" distributed system may also provide value for smaller-scale problems such as those required by log data. However, no evaluation method currently exists to adequately compare the capabilities of traditional (relational database) and modern NoSQL solutions for small-scale problems.

This research proposes a methodology to evaluate modern data storage and retrieval systems. While the methodology is intended to be generalizable to many data sources, a commercially-produced unmanned aircraft system served as a representative use case to test the methodology for aircraft log data. The research first defined the key characteristics of database technologies and used those characteristics to inform laboratory simulations emulating representative examples of modern database technologies (relational, key-value, columnar, document, and graph). Based on those results, twelve evaluation criteria were proposed to compare the relational and NoSQL database types. The Analytical Hierarchy Process was then used to combine literature findings, laboratory simulations, and user inputs to determine the most suitable database type for the log data use case. The study results demonstrate the efficacy of the proposed methodology.

Dedication

This work is dedicated to my mother who instilled integrity in me and always inspired me to do my best. I also wish to thank my wife and children for their consistent support, encouragement, and patience. Finally, I want to thank God for unwavering grace and love.

Acknowledgments

This research project would not have been possible without the support and guidance of my research committee. First and foremost, I want to thank my advisor, Dr. Brent Langhals, who asked me if I ever considered earning a PhD, helped me apply for this program, guided me towards and ultimately across the PhD finish line, and provided professional mentorship. Second, I want to thank Dr. Michael Grimaila for guiding me through my initial research effort, always pushing me to perform at a high level, and providing personal and professional mentorship and support. Next, I would also like to thank Dr. Douglas Hodson for providing peerless software engineering expertise, a multitude of creative ideas for research topics, questions, and directions, travel opportunities and companionship, unending support and encouragement, and great stories.

Next, I want to thank Dan Koorn, for his expertise, time, and patience. I also want to thank TSgt Charlie Loper for the same things.

Additionally, I would like to thank Dr. Timothy Lacey, Ryan Harris, and John Phillips for their support and computing resources for my research.

Also, I wish to thank the field study participants but since I promised to not reveal their information, no names will be provided.

Finally, I would like to thank Lt Col Logan Mailloux who provided encouragement, military mentorship and comradery, and ultimately friendship.

List of Figures

List of Tables

Table of Contents

A METHODOLOGY FOR EVALUATING RELATIONAL AND NOSQL DATABASES FOR SMALL-SCALE STORAGE AND RETRIEVAL

I. Introduction

Background

Many electronic systems capture events, state information, behaviors, and performance data as time-ordered records (Mittman & Dominick, 1973) (Schneier & Kelsey, 1999) (Sosnowski, Gawkowski, & Cabaj, 2011). Typically, log data is collected in flat files stored locally and/or forwarded to other locations (Johnson & Zwaenepoel, 1987) (Simache & Kaâniche, 2001) (Fan & Wang, 2010). Analysis of log data enables historic and current system status reporting, supports fault investigations, and provides insight for emerging system trends (Lin & Siewiorek, 1990) (Griffiths, Brito, Robbins, & Moline, 2009) (Ge, et al., 2010). Other uses include identifying patterns in system and user behavior, and event correlation (Peters, 1993) (Eick, Nelson, & Schmidt, 1994) (Yemini, Kliger, Mozes, Yemini, & Ohsie, 1996). Furthermore, record keeping is required by various regulatory bodies, such as the Federal Deposit Insurance Corporation and can be supported by log data collection and analysis (2014).

This research was motivated by the need to store and analyze a large amount of Unmanned Aircraft System (UAS) log data collected from multiple aircraft and generated at multiple distributed control stations. Log data collected from this system over a five-year period was made available for this research effort. Specifically, the end users desired a relational database to store and retrieve log data using a single computer system. However, due to changing system requirements and subsequent updates, the structure of the log data has repeatedly evolved thus complicating efforts to find a traditional relational database solution.

The end users initially believed the relational-based approach would be an intuitive solution for the problem. Since the codification of relational database model in 1970, organizations have primarily relied on relational databases to store and retrieve data of perceived value (Codd, 1970). Until recently, the relational model and Structure Query Language (SQL) have been the principal solutions for the majority of data storage and retrieval applications. However, by the mid-2000s, institutions began to realize that valuable data existed in diverse formats (blob, JSON, XML, etc.) and at a scale (petabyte or larger) not easily decomposable into exacting, pre-defined relational tables. Furthermore, with the extensive global reach of the Internet, organizations require accessible data independent from geography and often by millions of simultaneous users with a tremendously low tolerance for latency. Conventional relational databases were found trailing behind with respect to flexibility, scalability, and speed. Thus, a new generation of data storage and retrieval mechanisms emerged to address these shortfalls. These new technologies are commonly known as NoSQL or *Not Only SQL* systems (Schram & Anderson, 2012). Though the definition of the term *NoSQL* is inexact and often debated, for the purposes of this research, NoSQL is used to refer to all modern non-relational database types.

These NoSQL database types have developed along four general class lines: key-value, columnar, document, and graph (Han, E, Le, & Du, 2011) (Hecht & Jablonski, 2011) (Nayak, Poriya, & Poojary, 2013). Each one addressed issues of consistency, availability and partitionability, i.e. Brewer's CAP Theorem, while exploiting the benefits of accepting diverse data types and organizing storage/retrieval around the aggregate model rather than the enduring relational model (Fox & Brewer, 1999) (Sadalage & Fowler, 2013). In the NoSQL context, aggregates, represent a collection of data, or composite object, that is interacted with as a unit, equivalently setting the boundaries for ACID transactions (Sadalage & Fowler, 2013) (Evans,

2011) (Smith & Smith, 1977) (Elmasri & Navathe, 2016). Aggregates are fundamentally important because they define how data is stored and retrieved by the NoSQL databases. Implicitly, NoSQL databases must know more about the data to be stored and how it will be accessed. The aggregate model frees NoSQL databases from adhering to restrictive schemas and data structures typical of relational databases. As a result, NoSQL developers introduced database tools that are exceptionally fast, partitionable, and highly available.

In the rush to develop NoSQL databases to operate in large scale environments, distributed across commodity hardware and support a diverse array of data types, little attention of their use has been considered for traditional deployments such as single desktop/computer solutions. Thus, it is conceivable that some advantages may exist even in such a limited deployment. While some NoSQL advantages, like partition tolerance, are irrelevant in a single box implementation, others may remain.

Problem Statement

Given the relational database model was not always optimal to address the changing data structure occurring with UAS log data, a different database option should be considered. Though NoSQL databases provide solutions for storage and retrieval of data with varying structures and contents, these systems were designed for large-scale operations. Given that UAS log data end-users desired a single desktop solution, the storage and retrieval capabilities of the relational and NoSQL systems should be evaluated to determine which option is most suitable. However, no evaluation method existed to adequately compare capabilities of traditional relational databases and NoSQL solutions for such small-scale applications. The following research questions were devised to guide the study of database options in support of UAS log data in a single-box environment.

Research Questions

RQ 1. What are the defining characteristics of relational and modern non-relational, i.e. NoSQL, database systems?

This research question will investigate the aspects of modern, commonly-used database systems, i.e. storage and retrieval systems, and the associated terminology. The relational and aggregate models will be examined to discover any unique or commonly shared attributes. Common architecture designs will also be reviewed.

RQ 2. How can the characteristics of relational and NoSQL database types be evaluated to determine their suitability for single computer log data storage and retrieval systems?

This research question consider what criteria are necessary for comparing characteristics of modern data storage and retrieval systems to determine their suitability for deployment on a single system supporting transactions on log data. Additionally, an evaluation process will be considered to exercise the evaluation methodology.

Methodology

The research questions are answered through a combination of methods, procedures, and tools. A literature review was first conducted to outline the characteristics of modern data storage and retrieval technologies and propose effective criteria for the evaluation of each database option. Next, laboratory simulations were performed to quantitatively measure aspects of relational and NoSQL databases based upon the proposed criteria. Additionally, a field study with log data users provided real-world perspective on the appropriateness of the evaluation criterion. Finally, an evaluation using Analytic Hierarchy Process (AHP) combined the results of the lab simulation with the user provided inputs to identify the most suitable database system for the UAS log data.

Scope

The development and evaluation approach will be limited to a single-computer solution. This restriction provides an opportunity to explore the NoSQL database types in a way that has not be thoroughly discussed in academic or practitioner literature to date. Additionally, it provides a reasonable starting point for future research to extend where this research terminates.

While many modern data systems are well-suited for, tailored to, or may integrate data analysis, the analysis aspects of these systems will not be considered. The complexity of this aspect is largely dependent on the implementation of the data system. Additionally, the features of the evaluated data systems will be limited to data storage and retrieval. Features like user management, security, and others also vary widely between implementations. Thus, considering these characteristics of select implementations adds little value to a generalized evaluation approach for database systems.

Assumptions

The influence of cost and personnel are assumed to have some weight on the implementation of a log data storage and retrieval system. However, the author will assume that some funding would be available and that development personnel can be obtained to implement a solution. When the contrary is true, i.e. no funding and no personnel are available, then the problem becomes arguably less academic and more programmatic.

Preview

The remainder of this document is organized as follows. Chapter II provides a literature review outlining the characteristics of modern database technology. Additionally, the evaluation criteria are identified at the end of the chapter. In Chapter III, outlines a methodology to execute the

laboratory simulation and conduct the AHP using the results of the literature review, simulation, and field study. Next, in chapter IV, the results of the combined literature review, simulation, and field study are provided. Moreover, these results are used as inputs to the AHP to develop priorities using the criteria identified from Chapter II. These priorities are used to determine the most suitable database solution for the UAS use case. Finally, Chapter V discusses the findings of this research and provides recommendations for further study.

II. Literature Review

This section summarizes academic and industry literature to characterize modern data concepts, database technologies, and establish evaluation criteria for comparing five database types. First, a brief overview of modern data concepts is discussed to establish a context and background for relational and emerging NoSQL systems. Next, a brief definition of log data is provided. The second section describes the key characteristics of database systems. The last section details the evaluation criteria used to compare the database types for single system storage and retrieval applications.

Data Overview

Recently data has been characterized in terms of five V's: velocity, volume, variety, veracity, and value (Laney, 2001) (Elmasri & Navathe, 2016). Velocity entails how fast data is created, accumulated, ingested, and processed. Volume considers the size of the data of interest. Variety refers to the multitude of sources which generate data (Elmasri & Navathe, 2016). Data from these sources can be broadly grouped into five types: raw, cleansed, structured, semi-structured, and unstructured (John & Misra, 2017). Raw data has not been cleansed, filtered, or otherwise prepared for any particular analysis. In contrast, cleansed data has been readied for analysis (John & Misra, 2017) (Larose & Larose, 2015). Structured data logically fits into a table composed of columns and rows. Structured data may either be in a raw or cleansed state. Semi-structured data contains tags, keys, or markers which identify fields and records. Extensible Markup Language (XML) and JavaScript Object Notation (JSON) are two examples of semi-structured data. Finally, unstructured data is data that does not fall into the structured or semi-structured categories. Data in images, videos, and audio files are examples of unstructured data (John & Misra, 2017). Veracity describes

two features: the source's credibility and the data's suitability for the intended audience (Elmasri & Navathe, 2016). The data's value refers to the data's usefulness to its users (Jain, 2016).

The data lifecycle outlines the stages through which will pass during its existence. Various data lifecycle models exist (Ball, 2012) (USGS, 2017), but these models generally include acquisition, processing, analysis, storage, and disposal aspects (John & Misra, 2017) (USGS, 2017). Acquisition involves generating data from new sources or collecting data from a legacy or shared source (Boston University, 2017) (USGS, 2017). Processing includes calibration, extracting, integrating, preparation, organizing, structuring, transforming, and verifying data for ingestion or analysis (Larose & Larose, 2015) (USGS, 2017). Analysis actions include modeling, statistical analysis, quality assurance, pattern detection, interpretation, and hypothesis testing. Storage encompasses documenting, organizing preserving, structuring, and physical storage processes, e.g., ingestion, and activities which make the data accessible to the needs of users or applications. Disposal includes processes which review the need for data retention in addition to the activities which dispose of the data (USGS, 2017) (John & Misra, 2017). These concepts outline the characteristics of the data lifecycle.

Figure 1 presents a simplified view of the data lifecycle as a process. After data is acquired, it will be processed to prepare it for storage or analysis. After the analysis stage, data will flow to storage. From storage phase, data can return to the analysis phase or be disposed. Stakeholders may influence the data at any point in the lifecycle. Intuitively, most stakeholder interaction should occur during processing, analysis, and storage.

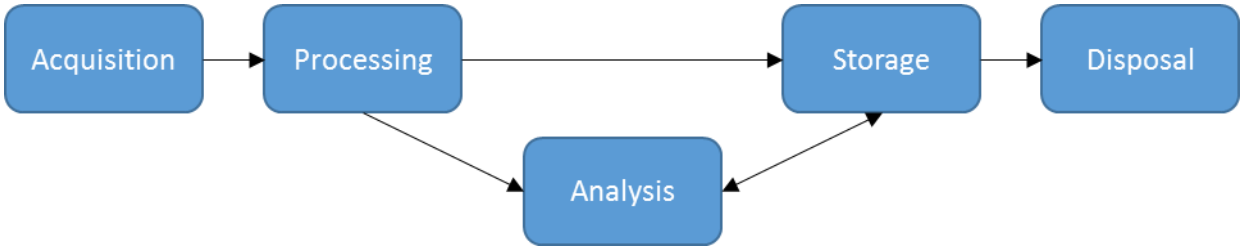

Figure 1. Simplified Data Lifecycle.

Traditionally, data would be structured during the processing phase and ingested during the storage phase. Only then would it be available for analysis. This data model has been characterized by Serra as "Structure-Ingest-Analyze." However, a new model, "Ingest-Analyze-Structure" is emerging with the development of modern technology. In this model, data can be ingested as raw, semi-structured, or unstructured data, analyzed, and then structured at a later time (Choosing technologies for a big data solution in the cloud, 2017).

Figure 2 presents the relationship between the 5 V's and the data lifecycle. Lines are drawn from each "V" data characteristic to the life cycle aspects which it influences.

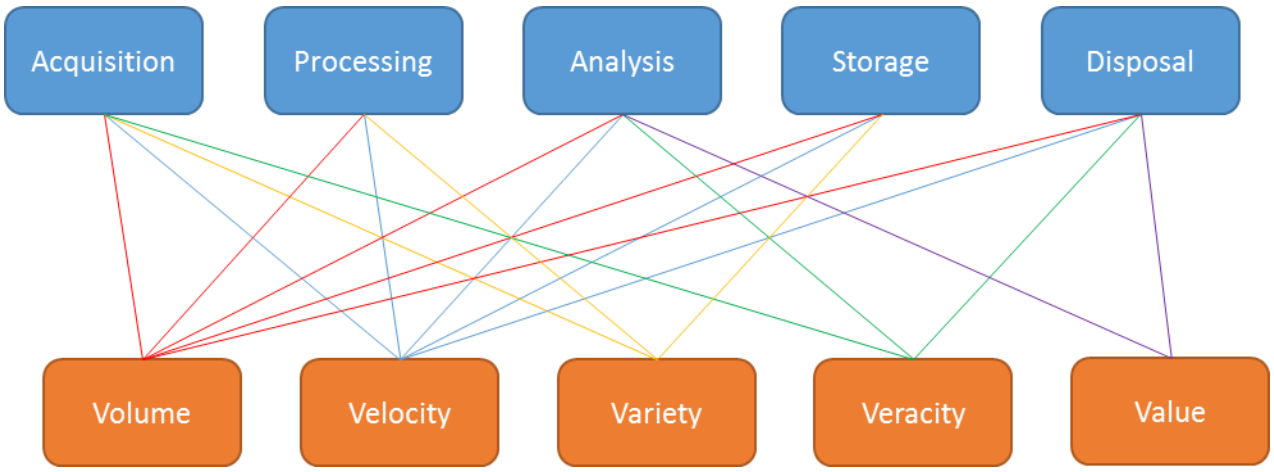

Figure 2. Graph of Relationship between Data Lifecycle and 5 V's.

Log Data Definition

In the scope of this research, log data is defined as "sequence of records" produced by an electronic system (Kreps, 2013). The term "sequence" implies the records have a relationship with time. This relationship will be explored in more detail at the end of the section. Additionally, an "electronic system" refers generally to all the electronic, computing, and information system elements composing a system as defined by IEEE (2015). Moreover, log data is also known as data logs or simply logs (PCMag, 2017) (IBM Knowledge Center, 2017) (Lee, Iyer, & Tang, 1991).

Log Data Storage and Retrieval

Typically, log data is collected in flat files stored locally and/or forwarded to other locations (Johnson & Zwaenepoel, 1987) (Simache & Kaâniche, 2001) (Fan & Wang, 2010). However, growing quantities of log data collected from distributed systems requires special attention to efficiently organize, store, access, and retrieve these records for analysis. Storing log data in a relational database (RDB) is one approach to meeting these needs. The widespread use of RDBs and support tools make them a candidate for these purposes (IBM, 2017) (Oracle Corporation, 2010). However, the semi-structured organization of the log files must be "wrangled," i.e., collected, aggregated, cleaned, and organized, into structured data before they can be imported (Joshi, 2016). Ideally, log data would be wrangled once to map data fields into RDB table attributes. However, changes to the log data structure periodically occur and affect the import process, database schema, and existing queries. The nature of systems may aggravate problems as changes may be introduced incrementally. This aspect requires use of multiple import processes; one process for each unique log file format must be developed and maintained to properly read and import the log data. Thus, effective solutions for log data management require a well-planned approach to support the needs of its stakeholders.

Summary

Log data is unique from other types of data in one significant aspect--log data is recorded as a sequence of events. Like other data, log data follows the data life cycle. Moreover, it is produced by a wide variety of sources and used for a multitude of purposes. Finally, due to its varying storage organization, i.e. being structured and semi-structured in some instances, a relational database solution may not necessarily be the most suitable solution for storage and retrieval. The next section describes database systems which provide the context for this work's evaluation of log data storage and retrieval options.

Database Systems

A review of available literature fails to identify a widely accepted definition of *data system architecture* or simply *database system*. The conventional definition of a database system includes application programs, Database Management System (DBMS) software, and database elements, is inadequate given modern data characteristics (Elmasri & Navathe, 2016, pp. 4-8). This definition is too limited for evolving data systems and fails to address the architecture aspects. Specifically, modern data systems are blurring the lines which traditionally separated business logic and data tiers (Kleppmann, 2017, p. 4). For example, Microsoft's SQL Server 2016 Enterprise Edition offers visualization, statistical analysis, and predictive analytics in a single package (Serra, Choosing technologies for a big data solution in the cloud, 2017). Furthermore, an architecture includes the "fundamental concepts or properties of a system in its environment embodied in it elements, relationships, and in the principles of its design and evolution" (ISO/IEC/IEEE, 2011). Thus, a definition which only includes the technical aspects is insufficient as an architecture. Therefore, a more thorough definition is needed. A better definition should consider the relationship between a system architecture and data.

The primary element of the data system architecture is the database system. As noted, the database system consists of one or more databases, DBMS software, and applications. A database is a shared collection of related data stored electronically, with an intended audience and purpose. The data collection is logically coherent, has meaning, and represents some real-world aspect such as business transactions. The DBMS supports the definition, construction, manipulation, and sharing of the database(s) among different users and applications. It is responsible for constructing, defining, manipulating, and sharing the databases. The applications provide users and external applications access point for the database. Specifically, the data system applications communicate with the DBMS software. Requests flow from the application through the DBMS to the database as queries. Queries are database interactions known as transactions. Transactions are processes which involve one or more database Create, Read, Update, or Delete (CRUD) operations. In a traditional database system, the database is optimized to support transactions on relatively small subset of data. Such systems are said to support Online Transaction Processing (OLTP) (Connolly & Begg, 2005) (Elmasri & Navathe, 2016) (Anderson, Lehnardt, & Slater, 2010).

A database system possesses four characteristics. First, it is self-describing. This property means the system contains the data collection and a description of the database's structure and constraints. Traditionally, this description resided in the database catalog in one or more schemas (Elmasri & Navathe, 2016). The second property of the database system is insulation or "program-data independence" (Elmasri & Navathe, 2016, p. 12). This property decouples the data from the applications enabling changes in one to be independent from changes in the other. Third, a database system should support multiple views which enable a subset or derivation of the data to be accessible though not explicitly stored in the database. The last property provides for data sharing and multiuser transactions (Elmasri & Navathe, 2016).

Figure 3 depicts the database system elements and their relationship to the traditional three-tier architecture. The applications provide for interactions between users and the data in the presentation tier. The DBMS controls access and may provide specialized business rules in the logic tier. The database or databases organize and store the data collection in the data tier (Elmasri & Navathe, 2016) (Microsoft, 2017) (IBM, 2017) (IBM, 2017).

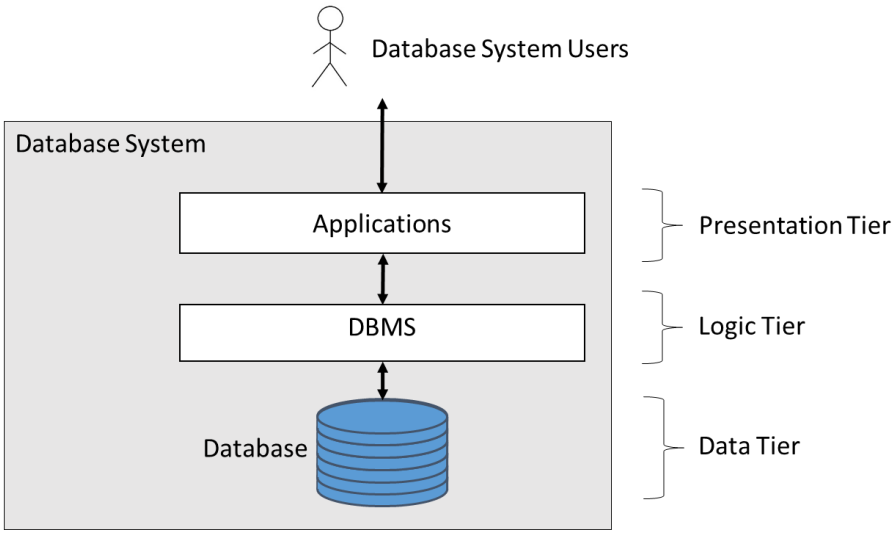

Figure 3. Simplified Traditional Three-Tier Database System Architecture.

This architecture was the basis for database system design since the 1970s (Codd, 1970). As stakeholder needs evolved, other database system elements were added. Additionally, various types of databases have been employed. The next two sections discuss the Data Warehouse (DW) and Data Lake (DL) system elements. An overview of the database types follows.

Data Warehouses

Data warehousing is defined as "subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions" (Inmon, 2002). In other words, data warehousing entails aggregating and cleaning data stored in multiple databases and multiple

formats to provide a single logical view of the collection. The process of aggregating, cleaning, integrating, (re-) structuring/reformatting, and transferring data from one or more sources into the data warehouse is known as ETL (Extract, Transform, and Load) (Elmasri & Navathe, 2016) (John & Misra, 2017) (SAS, 2017). The nonvolatile nature of data in a data warehouse means that data is generally not modified after ETL though it may be removed (Elmasri & Navathe, 2016).

Since the data warehouse is intended to support decisions, it is optimized for data retrieval by Decision Support Systems (DSS), Executive Information Systems (EIS), Management Information Systems (MIS), Online Analytical Processing (OLAP), or knowledge discovery applications. Information in data warehouses can be focused further into data marts which contain a subset of the DW's data. A data mart typically provides data warehouse analytical functionality but contains data geared towards one department or business line (Serra, Data Warehouse vs Data Mart, 2012). Both provide the means to explore warehoused data as a whole and contrasts with the OLTP optimization in databases (Fayyad & Stolorz, 1997) (Elmasri & Navathe, 2016).

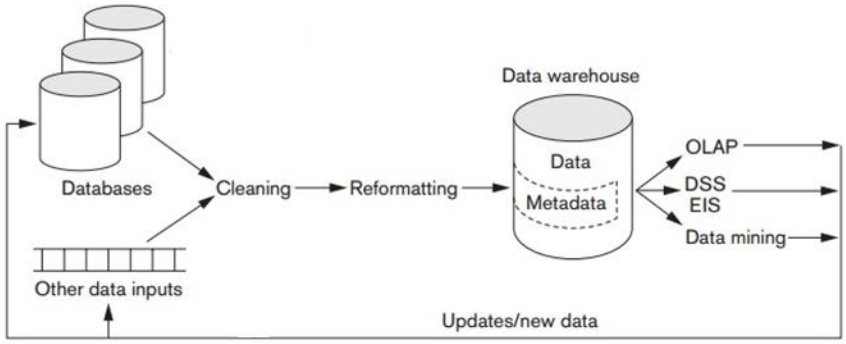

Figure 4. Data Warehouse Architecture Overview (Elmasri & Navathe, 2016, p. 1103)

Figure 4 presents an overview of the architecture for a data warehouse. Specifically, the architecture elements and their relationships are depicted. On the left side of the figure, multiple sources of data are shown. This data is cleansed and reformatted during ETL from these sources

to the data warehouse. Once in the data warehouse, the data can be analyzed by OLAP, DSS, EIS, MIS or knowledge discovery (data mining as shown) applications (Elmasri & Navathe, 2016).

Data Lake

The term "data lake" was coined by James Dixon as an improved approach over data marts and warehouses for storing, analyzing, processing, and delivering big data (2010) (John & Misra, 2017). Data lakes have evolved intended to serve as a centralized repository that can store, access, and retrieve "relevant" information (John & Misra, 2017). Unlike the other data system elements, data lakes can be populated with data in its raw format, including semi-structured and unstructured data. However, data lakes can also contain transformed or restructured data (John & Misra, 2017). Additionally, data lakes can be used independently from or in conjunction with the other data system elements (John & Misra, 2017) (Fowler, 2015) (Serra, Choosing technologies for a big data solution in the cloud, 2017) (Chen, Kazman, & Haziyev, 2016).

Figure 5 presents a data lake architecture which includes a lambda layer. In this figure, the data acquisition layer contains the applications to receive the incoming data. The message layer decouples the acquisition and ingestion layers using a guaranteed messaging paradigm. The data ingestion layer controls how the data flows into the lambda layer. The lambda layer enables batch (knowledge discovery, data mining, and ingestion) and real-time processing (ingestion and ad-hoc user queries) via its batch and speed layers. The server layer provides access and retrieval to users and other applications. Lastly the data storage layer contains and provides access to the stored data (John & Misra, 2017).

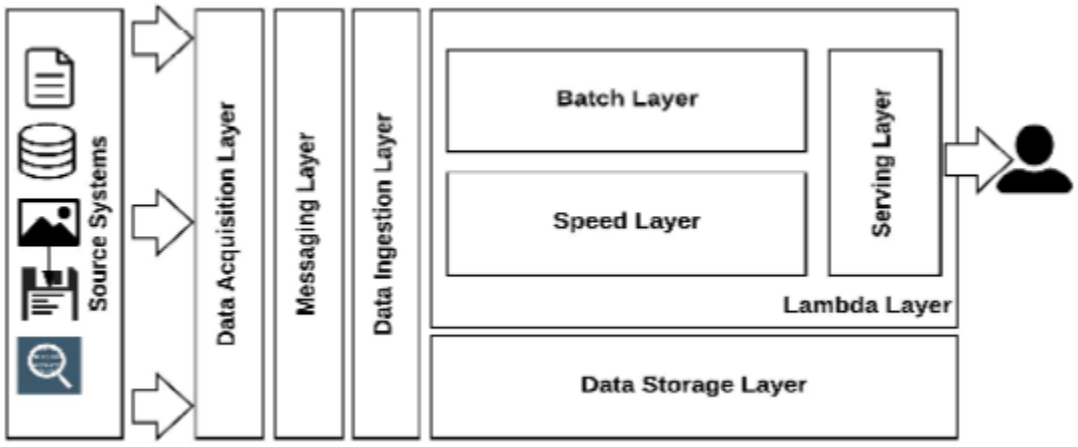

Figure 5. Data Lake Layers.

Database Types

Files and File Hierarchies.

Prior to the advent of relational databases, files and a hierarchy of flat files were used to organize and store data. Within a file, special characters were used to separate records and fields. For a file hierarchy, files which were organized by their location in the file system, indices or a both rather than an internal file structure (Sharpened Productions, 2017) (Codd, 1970). This technique created data access dependencies which Codd identified early in his work towards the relation model (1970).

Relational Databases

The relational data model has been a prominent technology since the 1970s (Codd, 1970). Systems using the relational data model have become known as SQL (Structured Query Language) systems because SQL is the standard interface to Relational Database Management Systems (RDBMSs) (Elmasri & Navathe, 2016).

RDBMSs organize data using Codd's relational model which is based on mathematical set theory and first-order predicate calculus (Elmasri & Navathe, 2016) (Codd, 1970) (Chamberlin & Boyce, 1976). Some key concepts from Codd's work include relations, tuples, domains, rows, relationships, attributes, and primary and foreign keys. He defined *relation* mathematically: given sets S_1, S_2, ..., S_n, which are not necessarily distinct, "R is a subset of the Cartesian product $S_1 \times S_2 \times ... \times S_n$" (A relational model of data for large shared data banks, 1970). A *tuple* is an ordered list of n-elements and likewise a *row* is simply an n-tuple which exists in R (Weisstein, n-tuple, 2016) (Codd, 1970). The set of permissible values on S_1, S_2, ... , S_n defines the *domain* for each set and each domain is considered an *attribute*. While relations are domain-ordered, Codd explained relationships are not dependent on this order. Furthermore, a *primary key* is a domain or domain combination containing values that uniquely identify every row of a given relation. Similarly, a *foreign key* is a domain or domain combination in R that is not a primary key of R, but is the primary key of another relation, S (A relational model of data for large shared data banks, 1970).

Normalization is another aspect of the relational model and was Codd's preferred method for organizing data (Codd, 1970). The normalization process examines functional dependencies and primary keys of a relational database. Its goal is to minimize both redundancy and update anomalies that could occur during insertion, deletion, and modification operations that could affect data consistency (Elmasri & Navathe, 2016).

Most relational systems require a schema to be defined before data is loaded into the database. Such a schema describes tables, constraints, keys, and relationships to provide a structure for the database. Some schemas may also provide for user roles and additional *views* of the data. Whereas a table is closely related to the physical storage of the data, a view is a logical structure providing access to various data elements which already exist in tables. Additionally, a schema itself provides

a detailed, logical outline of or structure for the entire the database. Once a schema is established, it is expected to undergo few changes (Elmasri & Navathe, 2016) (Hoffer, Venkataraman, & Topi, 2016) (Bradbury, 2017).

NoSQL Databases

In this research the term, *NoSQL databases,* refers to all modern non-relational database types. These NoSQL database types encompass a collection of systems including key-value, document, column, and graph data models. These systems represent a growing area of the database market (Elmasri & Navathe, 2016) (Connolly & Begg, 2005) (Sullivan, 2015) (Redmond & Wilson, 2012) (Marz & Warren, 2015) (solid IT gmbh, 2016).

Aggregate concept

One defining characteristic for NoSQL data stores is the use of aggregate stores or an aggregate-oriented model. The aggregate concept is a helpful way to contrast NoSQL database types with each other as well as with the relational databases and implies a certain level of knowledge exists regarding what data is stored and how it will be retrieved. An aggregate is formally defined as a composite data object that is considered the atomic unit for Create, Read, Update, and Delete (CRUD) operations. This concept originated with databases and software development in the 1970s and is also related to Evans's recent work with domain-driven design. In the NoSQL context, aggregates may vary widely in size and composition, ranging from individual binary values representing status flags to MPEG video files and their associated metadata. Treating data as aggregates enables data stores to take advantage of locality and de-normalization to improve data retrieval performance (Smith & Smith, 1977) (Evans, 2011) (Sadalage & Fowler, 2013) (Elmasri & Navathe, 2016) (Denning, 2005).

Empowering the aggregate model concept is the ability of NoSQL to accept data without any prerequisite data modeling, unlike relational databases where a schema or predefined logical view of the database must exist before data can be imported (Elmasri & Navathe, 2016) (Bradbury, 2017). Thus, the NoSQL database structure, i.e. the physical aggregate, emerges as data is added (Robinson, Webber, & Eifrem, 2015) (Hoffer, Venkataraman, & Topi, 2016) (Bracket & Kempe, 2013). Additionally, NoSQL databases excel at storing and retrieving semi-structured and unstructured data which do not fit well inside a traditional RDB. Unstructured data may include plain text format or multimedia, while semi-structured data may be organized into JSON or CSV formats. Both JSON and CSV formats impose some structure on data, but the contents can vary. Semi-structured data is also referred to as having a hybrid structure. RDBs primarily operate on structured data, which is data that is easily organized into a rectangular table and normalized. In contrast, NoSQL databases can store and retrieve all data types efficiently (Leavitt, 2010) (Cattell, 2011) (Hecht & Jablonski, 2011) (Joshi, 2016). The following sections discuss the four general classes of NoSQL databases and their defining characteristics.

Key-value Databases

Key-value (KV) data models store and retrieve data as key-value pairs. The key is a unique identifier and the value is the data associated with the key. These pairs are similar to maps, dictionaries, and associative arrays which use non-integer indexing to organize data. Key-value databases do not require a defined set of key or value types to be used (Sullivan, 2015).

In this data model, each stored value constitutes the complete aggregate. Additionally, aggregates are isolated and independent from each other. Thus, no relationships are stored in this data model. Furthermore, few limitations are placed on what data types can be stored as values. Values may contain strings, integers, maps, lists, sets, hashes, queues, Binary Large OBjects

(BLOBs), or a composite object of these types (Gudivada, Rao, & Raghavan, 2014) (Josuttis, 2012) (Hecht & Jablonski, 2011) (Sadalage & Fowler, 2013).

KV databases treat aggregates as opaque atomic units once they are imported. "Opaque" means the database has little knowledge about what comprises the value stored or how it is structured. However, this feature provides for great flexibility in storage, simplicity for querying, and shifts responsibility for data integrity outside of the database. Additionally, KV databases generally do not include a complex query processor. CRUD operations are accomplished using put, get, and delete operations. Thus, complex queries must be handled at the application layer outside the database (Hecht & Jablonski, 2011).

Document Databases

The document model is in many ways similar to the KV model. Document models organize and store data in a document structure consisting of a set of key-value pairs. More formally, a document is a self-describing, i.e., key-value pairs, hierarchical tree data structure consisting of scalar values, maps, lists, other documents, and collections. A collection is a group of documents and often pertains to a particular subject entity. Furthermore, document-based databases are free from a requisite schema and instead infer information from the document structure itself (Sullivan, 2015).

The aggregate is the document in this model. The inclusion of keys in the aggregate provides the self-describing aspect of this object (Cattell, 2011) (Sadalage & Fowler, 2013) (Sullivan, 2015). Like the KV model, most data types can be stored in a document model including Boolean values, integers, arrays, strings, dates, and BLOBs among others. Additionally, document models employ a unique identifier to distinguish individual, top-level documents. While a document is

comparable to a row in a relational database, it does not natively store relationships between documents with the exception of nested documents (Cattell, 2011) (Sadalage & Fowler, 2013) (Sullivan, 2015).

A few more aspects of the document model differ from the KV model. Document models provide aggregate transparency which enables access during read, update and delete operations to attributes/data elements stored within an aggregate. This characteristic is unlike the opaque nature of KV models. Additionally, document stores typically include a query processor that can perform complex queries such as searching for a range of values, accessing keys within a document, or handling conditional query statements like those common to SQL. Yet, like a KV model, responsibility for data integrity and any relational consistency is placed outside the database itself. Furthermore, document models often include indexing to speed up searches. Lastly, attributes can be added to existing documents (Cattell, 2011) (Sadalage & Fowler, 2013) (Sullivan, 2015).

Documents are the lowest level of components defined and are an ordered set of key-value pairs. Dynamic schemas define collections. That is, documents organized in a particular collection do not have to "look" the same. Just as documents are grouped into collections, a set of collections comprises a database. Several databases can exist within a single instance of a document database. As relational database analogs, documents are comparable to rows and collections are akin to tables (Chodorow, 2013).

Column Databases

The column family, or column-oriented, model organizes data into a multidimensional map based of the Decomposition Storage Model (DSM). Originally the DSM organized data into columns which were associated by a unique identifier known as a surrogate. In this model, the

column is the basic storage unit and composed of a name and a value, much like a key-value pair. Columns may be grouped together as column families. Rows are composed of a unique key and one or more columns and/or column families (Copeland & Khoshafian, 1985) (Abadi, Boncz, & Harizopoulos, Column-oriented database systems, 2009) (Hecht & Jablonski, 2011) (Sadalage & Fowler, 2013) (Sullivan, 2015).

Though the terminology is similar to the relational model, a row in this model is actually a two-level map. Figure 6 presents an example consisting of two rows, to illustrate the two-level map properties of the column family database. The first row, row key "1235," contains a single column consisting of a column key designated as "name" and its value is "Grad Student A." The second row, row key "1236," contains a slightly more complex column family. In the column family, there are four keys, "firstAuthorName," "secondAuthorName," "thirdAuthorName," and "fourthAuthorName" and four associated values, "Grad Student A," "Professor X," "Professor Y," and "Professor Z."

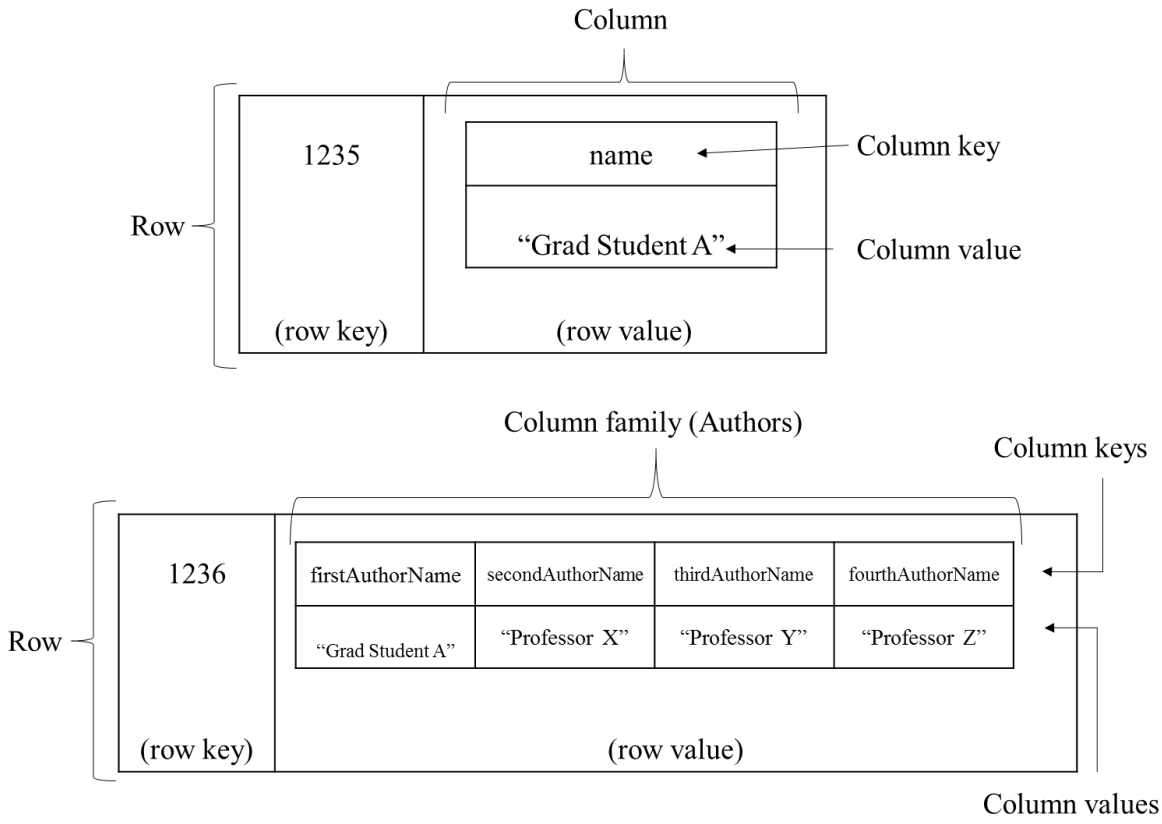

Figure 6. Examples of rows in a columnar database.

In this model, the row value is the aggregate. Additionally, column family models provide aggregate transparency, like the document model, to provide access to individual columns within the aggregate. Furthermore, columns can be added, updated, or excluded from rows without updating a predefined schema. However, column families usually must be defined before they are used. Finally, column family databases often include a query processor to facilitate searching and retrieval (Abadi D. J., 2008) (Hecht & Jablonski, 2011) (Abadi, Boncz, Harizopoulos, Idreos, & Madden, 2013) (Sadalage & Fowler, 2013) (Sullivan, 2015).

Typically columnar databases do not require a schema in the relational database sense, but do require a keyspace, row keys, columns, and column families to be defined explicitly. A keyspace

is considered the highest level structure in a column database because it contains all other data structures used in a particular DB. In fact, Sullivan stated a keyspace is analogous to a relational schema. Row keys are like primary keys because they are used to uniquely identify rows. A column structure stores a value and can consist of a column name, a time or version stamp, and the value itself. Finally, column families are a related group of columns and are like relational database tables because they store multiple columns and rows (2015).

Graph Databases

Graph database systems do not require a formal schema; however, these systems do require a structure (Robinson, Webber, & Eifrem, 2015, pp. 106, 109). Graphs consist of two elements: vertices and edges. A vertex or node is a particular point or location where two or more lines or edges intersect (Weisstein, Vertex, 2016). Likewise, the line connecting two unordered nodes is an edge. In a directed graph, the node order is relevant (Weisstein, Graph Edge, 2016).

Property graph models are common implementations of the more general graph model. Property graph models store and retrieve data using the two primary modeling objects: nodes and edges. A node represents an entity and stores any attributes as properties. Likewise, an edge represents a relationship between one or two nodes. Edges have an associated direction between nodes and may also include properties. Properties for either nodes are edges are stored as key-value pairs. (Hecht & Jablonski, 2011) (Sadalage & Fowler, 2013) (Robinson, Webber, & Eifrem, 2015) (NOSQL Databases, 2018).

Graph models share many characteristics with other NoSQL data models but have some unique traits. Graph models support most primitive data types such as Boolean, byte, short, int, long, float, double, and char types. Naturally, arrays of these primitives are also permitted. Storage of BLOBs

are permitted but are not as well suited for the graph model. Graph models are said to be relationship oriented and most appropriate for heavily linked data. Additionally, they are unique from the other NoSQL data models because there are two elements (nodes and edges) which can comprise an aggregate. That is, data can be stored on either a node or an edge in this model to be considered an aggregate. Furthermore, nodes and edges may be defined in a schema or added as necessary (Hecht & Jablonski, 2011) (Sadalage & Fowler, 2013) (Robinson, Webber, & Eifrem, 2015).

Graph models are considered to represent relationships more naturally than other data models. Graphs can overcome the impedance mismatch commonly found in relational models of data objects. The object-relational impedance mismatch describes the difference between the logical data model and the tabular-based relational model resulting in relational databases. This mismatch has long created confusion between technical and business domains (Hecht & Jablonski, 2011) (Robinson, Webber, & Eifrem, 2015).

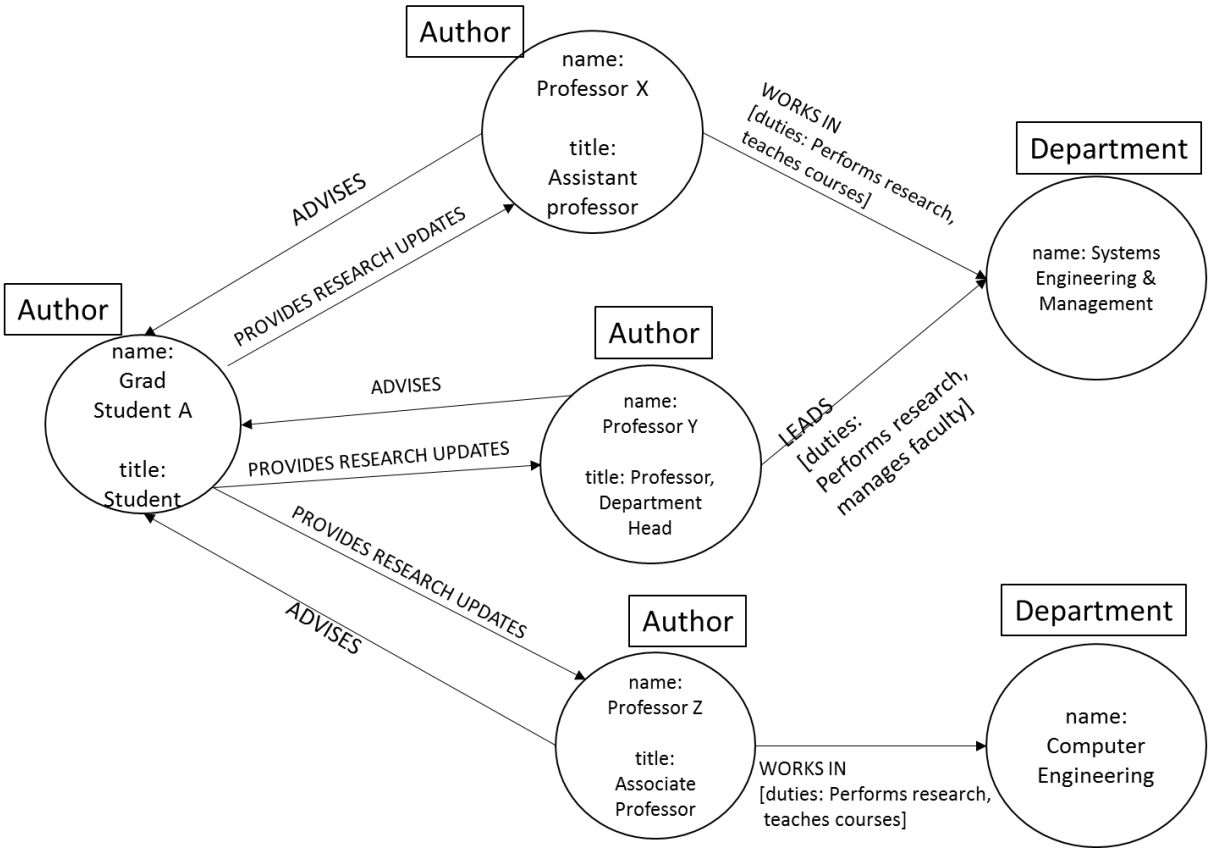

Figure 7. Example of a property graph model.

Figure 7 presents an example of a property graph depicting a few relationships between the authors and their associated departments. In this figure, two kinds of nodes, "Author" and "Department," and nine relationships are displayed. Each "Author" node has two properties: "name" and "title." Similarly, the "Department" nodes contain one property called "name." The values for each property are shown in the diagram. Relationships are interpreted by starting with one node and following a directed edge to its related node. For example, the Author named "Professor X" ADVISES the Author named "Grad Student A." Additionally, the Author named "Professor X" WORKS IN the Department named "Systems Engineering & Management". The Author's "duties" in this relationship are to "Perform[s] research" and "teach[es] courses." The other relationships are interpreted using the same method.

Retrieval and Transactions

Data retrieval from a DBMS typically involves a *transaction* which is a process involving multiple steps all of which must be completed for the transaction to be considered successful (Sullivan, 2015) (Gray, 1981). Five concepts describe and provide context to transactions in DBMSs: queries, the CAP theorem and its extension, PACELC, ACID, and BASE. The ACID principle pertains primarily to RDBMSs while the BASE concept generally applies to NoSQL systems (Redmond & Wilson, 2012) (Sullivan, 2015) (Brewer, 2012). In contrast, the CAP and PACELC theorems apply to any distributed database system (Fox & Brewer, 1999) (Abadi D. J., Consistency tradeoffs in modern distributed database system design, 2012). These concepts will be described in detail in the following sections.

ACID

The ACID principle involves four properties concerning a transaction: Atomicity, Consistency, Isolation, and Durability (Haerder & Reuter, 1983). An atomic transaction is considered an indivisible unit. Thus, all transaction steps must be completed or none of them will complete (Sullivan, 2015). A consistent transaction results in a coherent and logical data view (Sullivan, 2015). Isolation ensures that transaction operations are not apparent to users until the transaction completes. Lastly, a durable transaction ensures its results persist given a subsequent malfunction such as power failure or server crash (Haerder & Reuter, 1983) (Sullivan, 2015) (Redmond & Wilson, 2012). Such transactions undergo a two-phase commit in which databases agree to perform the operation and then perform the operation once agreement is communicated (Pritchett, 2008). Again, most relational database transactions comply with the ACID principle and most NoSQL systems only provide ACID-like functionality at the aggregate level (Sadalage & Fowler, 2013).

BASE

The alternative to strict ACID-compliance transactions are those which support Basically Available, Soft state, and Eventually consistent (BASE) transactions. Basically available indicates the DBMS will usually be accessible by authorized users and applications. Soft state suggest that databases on different servers will not always contain matching copies of data. Eventually consistent entails that data systems will be inconsistent across servers some of the time. (Robinson, Webber, & Eifrem, 2015, p. 195) (Pritchett, 2008) (Brewer, 2012) (Vogels, 2009).

Queries

A query is an action performed on a database, using a language such as SQL, to access a specific set of data (Codd, 1970) (Baker, 2011). In a relational database, a query will often involve a join operation to de-normalize data across tables (Codd, 1970). Both NoSQL and relational system employ queries to address data and perform searches and transactions involving Create, Read, Update, and Delete (CRUD) operations. Transactions perform a single CRUD operation on a single atomic unit (Codd, 1970) (Chamberlin, et al., 1976) (Chodorow, 2013) (Carpenter & Hewitt, 2016) (White, 2015) (Anderson, Lehnardt, & Slater, 2010) (Redmond & Wilson, 2012).

CAP and PACELC

Brewer's theorem, or the Consistency, Availability, and Partition-resilience (CAP) theorem (Gilbert & Lynch, 2002) (Fox & Brewer, 1999), defines three properties of a distributed database system: consistency, availability, and partition tolerance. Consistency means each server has the same version/copy of data viewable for transactions. This consistency is different from ACID consistency. Availability is the ability to provide a response to a transaction request. Partition resilience (also tolerance or protection) is the condition where a system will remain operational if the network connecting two or more database nodes fails, i.e., a network failure

results in a partition. The CAP theorem states a distributed system can achieve at most two of the three principles (Fox & Brewer, 1999) (Elmasri & Navathe, 2016) (Sullivan, 2015).

This binary understanding of the CAP theorem has been criticized by even Brewer himself (2012) and has contributed to its PACELC ("pass-elk") extension. PACELC can be understood as follows: if a network partition (P) occurs, then designers must balance availability (A) and consistency (C); else (E) no partitions exist, designers must balance latency (L) and consistency (C) (Abadi D. J., Consistency tradeoffs in modern distributed database system design, 2012).

Scalability

Scalability refers to a database system's ability to increase its capacity while functioning normally without interference or minimal performance degradation. Vertical scalability pertains to increasing the resources of a single server, whereas horizontal scaling means to increase the number of available servers. Horizontal scalability also leads to distributed database (DDB) system design. Three concepts outline the features necessary to design a DDB system: fragmentation, replication, and allocation (Elmasri & Navathe, 2016, pp. 842, 845, 847) (Carpenter & Hewitt, 2016, p. 19). In general terms, scalability is an inherent feature of many NoSQL systems and a capability that can be achieved with mixed success for RDBMSs. This section will compare both types of scaling, summarize fragmentation, replication, and allocation, and contrast the capabilities of relational and NoSQL systems to explain this claim.

Additionally, it is appropriate to define Big Data during this discussion. Big Data definitions typically include properties known as the "three Vs": volume, velocity, and variety. The amount or volume of this data can extend into the petabyte range (1K Terabytes). Velocity is simply a measure of how fast the data grows and typically ranges from real-time to daily time frames.

Variety refers to both data sources and formats (Amazon Web Services, 2016). Other "Vs," such as variability, veracity, visualization, and value, have been included as well (Allen, Jankowski, & Pathirana, 2015) (McNulty, 2014) (Chen, Shiwen, & Liu, 2014).

Vertical vs. Horizontal Scalability

The inherent limitations of vertical scaling should be addressed. Consider a server upgraded to its maximum capacity in terms of processing power, memory, and disk storage. Currently, some affordable servers offer memory capacity in the single terabyte (TB) range and storage capacity around 100 TB (Athow, 2016). If the volume of data it needs to process and store exceeds this capacity, then the system has reached an upper limit and cannot operate effectively. Now consider that in 2013, an estimate of the digital universe's size exceeded 4 zettabytes (4 billion TB) with an expectation it would increase by an order of magnitude to over 40 zettabytes by 2020 (IDC, 2014). The New York Stock Exchange creates 4-5 terabytes of data daily, Facebook's photo data grows by 7 petabytes monthly, and the Internet Archive hosts approximately 18 petabytes (White, 2015, p. 3). The suggestion that a single server is inadequate does not appear unfounded. These facts underscore the current (or pending) limitations of vertical scaling.

Reflecting on the numbers cited above, the notion of horizontally scaling to thousands of systems may seem improbable. Yet in 2013, Microsoft claimed to have one million servers which CEO Steve Ballmer claimed was less than Google's but larger than Amazon's (Anthony, 2013). Hosting thousands of servers now seems to exist within the realm of the possible. Additionally, scaling in this manner naturally supports a larger user base than a single, vertically scaled system—the Internet itself is proof of this claim.

Fragmentation

Fragmentation involves breaking the database into logical units. Two types of fragmentation are common: horizontal (or sharding) and vertical. A shard or horizontal fragment of a table is a subset of its rows. Shards are commonly organized to segment the data by feature or function, by a hashing algorithm, or through a look-up table. An additional concept related to shards is that of a "share-nothing" architecture in which no centralized state exists between shards and each server operates independently. We will see this idea opposes the concept of replication. In contrast to sharding, vertical fragmentation divides tables by columns. Hybrid or mixed fragmentation occurs when a combination of sharding and vertical fragmentation is used (Carpenter & Hewitt, 2016) (Elmasri & Navathe, 2016).

Replication and Allocation

The replication concept involves copying all or part of the database across multiple servers. A fully replicated distributed database will have a full copy of the database on every server in the system. Alternatively, some fragments are replicated. Replication can improve availability by ensuring some nodes are available when others go offline. Allocation is the process of assigning replications to particular nodes in the system. When fragments are disjoint, i.e., every fragment is stored at only one site except for primary keys, it is known as nonredundant allocation (Elmasri & Navathe, 2016, pp. 849-850).

Concepts of master, slave, cascading replication, and concurrency control are included with replication models. Concurrency control is used to manage copies of the data, failure of nodes or communication links. Certain implementations, such as those supporting two-phase commits, must also manage distributed commits (Carpenter & Hewitt, 2016, p. 12). Additionally, deadlock may occur amongst distributed systems and requires management. A three-phase commit protocol is one solution to the distributed commit and deadlock problems (Elmasri & Navathe, 2016, p. 858).

Some systems may employ a master and slave relationship between servers to facilitate the replication process. Additionally, some implementations, such as PostgreSQL, use cascading replication to propagate replication from slave to slave (Elmasri & Navathe, 2016, p. 854) (Obe & Hsu, 2015, pp. 181-2).

Scaling in ACID vs. BASE System

NoSQL BASE systems are typically designed with scalability in mind. Key-value databases employ methods of replication (Sullivan, 2015). Key-value and column DBMSs tend not to shard their data (Robinson, Webber, & Eifrem, 2015). Document database supports replication of its databases and can be initiated to operate manually or continuously. In this way, it supports read request scaling by providing more than one database to provide load balancing and higher availability. Additionally, some document databases, such as CouchDB, provide a means to shard and distribute a logical database across servers (Anderson, Lehnardt, & Slater, 2010). Likewise, MongoDB was designed for horizontal scaling and automatically balances data loads across servers. When new machines are added, MongoDB will determine how to spread existing data to them. Additionally, MongoDB includes autosharding, a process which partitions data across various servers (Chodorow, 2013). Graph databases are an anomaly when it comes to NoSQL scaling because they typically do not scale horizontally very well. However, even an exception exists within this exception: Titan is designed for horizontal scaling (Sullivan, 2015).

After reviewing the horizontal scalability concepts, one can speculate how systems strongly adhering to the ACID transaction principles will experience more complexities and performance degradation than systems in which BASE principles are expected (Dash, 2013). For example, consider the latency involved with a two- or three-phase commit across 1000s of nodes. If each system operates independently, then each system has to wait for, receive, and process a commit

message from the other systems in order to complete the commit. Even systems with more efficient master and slave relationships or nonredundant allocation will have similar issues of complexity compared to those designed to scale out. In short, RDBMS, ACID-based systems have struggled to meet the demands of Big Data scaling (Sullivan, 2015).

Single Desktop System Database Evaluation Criteria

This section describes criteria by which the characteristics of relational and the four NoSQL database types can be examined. Figure 8 displays the 12 criteria organized as a hierarchy. In this figure, the top node identifies the system of interest—a data storage and retrieval system. The next level of nodes provides a means to logically group the 12 criteria. Specifically, storage, retrieval, and aggregate properties partition the criteria.

Overall, the 12 criteria were derived from relational and NoSQL database traits that remain relevant for a single box environment. Thus, many of the typical NoSQL characteristics associated with large scale data and applications are excluded. For example, horizontal scalability was not considered because this characteristic pertains to large-scale, distributed systems only. Additionally, the identified criteria focus on data storage and retrieval operations and mechanisms available within the databases themselves. That is, the database type does not rely on an external application to provide the feature.

For this discussion, an *aggregate* is a collection of all the variables of interest sampled at a given moment in time. “Of interest” refers to the variables intended to be stored, retrieved, or updated in the DBMS. Likewise, an *element* would be a single sample of a variable, such as engine speed, altitude, or heading. Furthermore, a *transaction* is a CRUD operation performed on a single aggregate and a *query* is a set of transactions executed on one or more aggregates and/or elements.

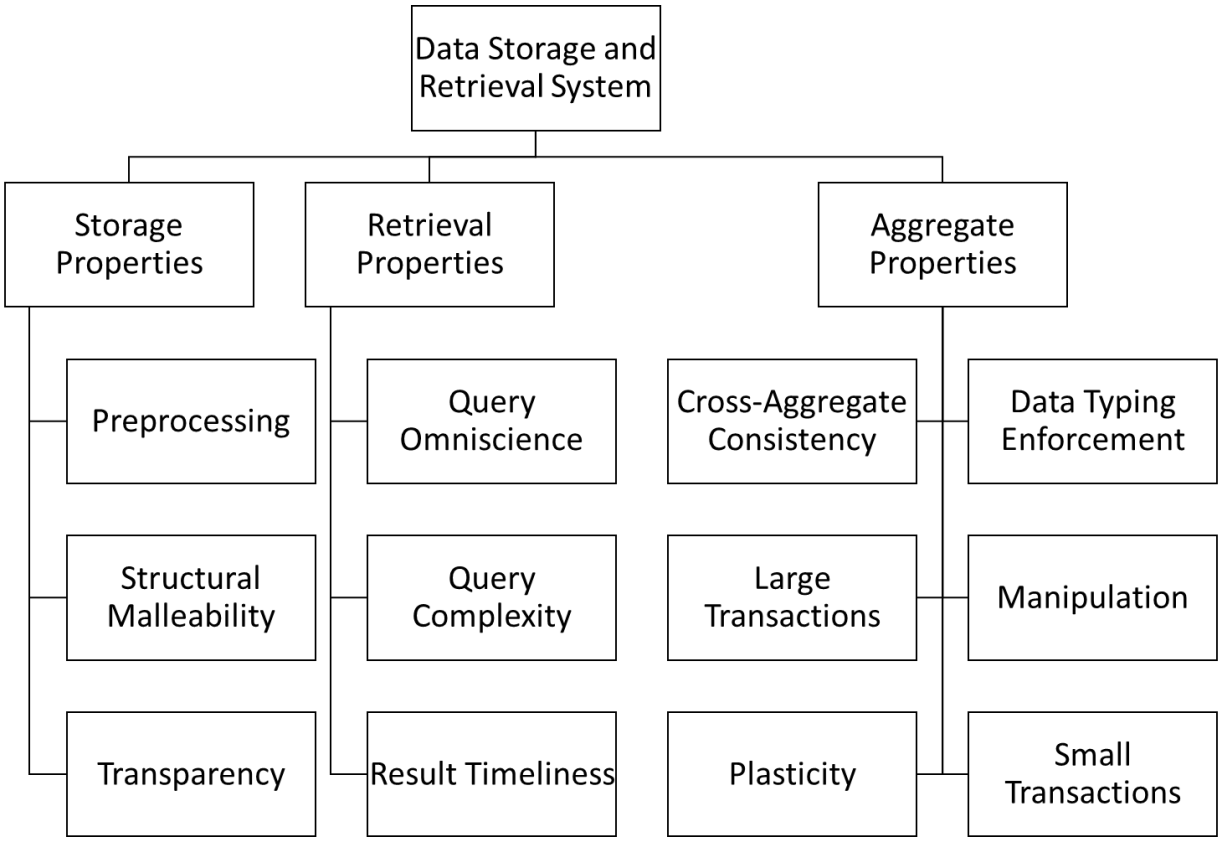

Figure 8. Evaluation Criteria

The proposed evaluation criteria are logically grouped by the type of operations they influence. First, *Storage Properties* refers to characteristics of NoSQL databases that affect either extent to which data must be manipulated prior to storage or the ability to create/delete aggregates and then view them once created. The second characteristic, *Retrieval Properties*, captures how well each database type can return stored data. The power of the NoSQL aggregate model stems from storing related data physically together, allowing efficient retrieval. Implicitly then, certain aspects of the stored data and more importantly how it is expected to be returned must be known a priori. For retrieval operations, the characteristics of interest include query complexity, the amount of a priori knowledge known, and the timelines required for query returns. The last set evaluation criteria

involve properties of the aggregates generated by each NoSQL database type. They include issues of consistency, data typing, ability to handle varied transaction size, and ability to update and manipulate each aggregate. Additionally, this set includes criteria involving properties which cannot be exclusively categorized as storage or retrieval that is some of these criteria incorporate aspects related to both storage and retrieval.

To provide clarifying examples of how the criteria are to be used, a database system designed for Unmanned Aircraft Systems (UAS) log data is discussed. For this UAS, log data is generated by the periodic sampling of various subsystem status variables. The following sections discuss each of these criteria in greater detail and outline the relative importance for each.

Storage Properties

The first storage property is *preprocessing* and includes all preprocessing and associated operations required to import/load data into the DBMS. Examples of preprocessing may include: deciding/selecting which variables to store in the DBMS, filtering the selected variables from the raw data, transforming filtered variables into a format suitable for loading, and loading transformed data into the DBMS. When evaluating this criterion, one should consider the ability to perform preparation work before data can be loaded into the DBMS. If unwilling (or unable), then this criterion is important because the DBMS is expected/required to facilitate data preparation, preprocessing, or loading of raw data. However, if the user is willing and able to handle the preprocessing, then this criterion is less important because the DBMS is not expected/required to facilitate this process.

The second storage property is *structural malleability* which refers to the DBMS's ability to add/remove "types" of aggregates to and from the DBMS. For example, aggregate types for a UAS

database could include: engine subsystem, pilot inputs, aircraft telemetry, and others. If, perhaps, a new sensor system was added to the UAS, a new aggregate type might need to be added to the DBMS to store and organize data for this system. When considering this criterion for a UAS database, it would be important to think about the likelihood a new subsystem (thus new aggregate) would be added to or removed from the aircraft. If likely, then this criterion is important because a solution to support aggregate type changes is needed. If not, then this criterion is less important.

The final storage property is *transparency.* In this case, transparency describes the DBMS's ability to store aggregates such that individual elements within the aggregate can be viewed and retrieved during read transactions. For example, assume a DBMS is designed using an aggregate model that contains all variables for a sample in time. Thus, engine speed, altitude, landing gear, tail number, timing information, etc. are stored in each aggregate. When a DBMS supports transparency, it is possible to search for and retrieve engine speed from an aggregate. Without transparency, searches are limited to retrieving the entire aggregate, not the specific value for engine speed. Users should consider if they care about retrieving individual elements, such as engine speed, from the data. If so, then this criterion is important because there is a need to access individual data elements within aggregates. If not, then this criterion is less important.

Retrieval Properties

The first retrieval property criterion is *query complexity* which describes a DBMS's ability to perform both simple and complex queries. A simple query retrieves data using only a unique identifier or a range of identifiers and/or values. Complex queries involve additional conditions enforced on the operations. For example, a simple query might ask 1) if the engine speed ever exceeded 5500 RPM or 2) if the engine speed exceeded 5500 RPM on the mission flown today.

An example complex query, however, may ask 1) if the engine fan drew more than 33 Amps while the cooling fan was set to auto or 2) what is the average coolant temperature during a mission for each aircraft. Users should consider whether they need the database to perform sorting, grouping, mathematical calculations, and conditional searches on their data sets. If yes, query complexity is important. If not, query complexity is less important.

The second retrieval criterion is *query omniscience* and it refers to the degree to which the complete set of possible queries is known by the user before system is implemented. For example, if the user knows every question that will ever be asked about the data and does not anticipate unexpected events or circumstances will require unique investigations, then this criterion is important because the user will not need the DBMS to support additional queries/questions in the future. However, if the possibility exists for new unanticipated queries to be developed or for existing queries to be changed, then this criterion is not as important.

Finally, *result timeliness* is the last retrieval criterion. Result timeliness refers to how quickly the results are provided to the user after a request is made. For example, a query is described, run, and produces results within 10 seconds of starting to run. If the user expects results within seconds or less, then result timeliness is important. However, if the user is willing to wait minutes or longer for a result, then result timeliness is less important.

Aggregate Properties

The first evaluation criterion for aggregate properties is *cross-aggregate consistency*. This criterion refers to the DBMS's ability to perform cascading updates to data and relationships. Assume a UAS was flown on multiple distinct flights and the log data and tail number for each flight is loaded into a DBMS as part of several different aggregates. Additionally, the DBMS stores

the relationship between the UAS and each flight as part of these different aggregates. Under such a scenario, queries can determine which flights were flown by a particular UAS as well as any associated log data by looking for tail numbers across the aggregates. Queries can also retrieve/update properties about relationships such as the start/end date and time of all missions flown by a UAS using its unique tail number. Later, someone realizes the wrong tail number was recorded for a given flight, resulting in an update which affects all aggregates containing data from the affected UAS. Cross-aggregate consistency involves the process responsible for updating each aggregate with the appropriate tail number. When updates to stored data are required, cross-aggregate consistency is important, because the DBMS is expected to enforce consistency among data and stored relationships. If the user can tolerate some inconsistencies or cross aggregate consistency can be managed in another way, then cross aggregate consistency is less important.

Next, *data typing enforcement* describes the extent a DBMS applies schema enforcement for data typing during transactions. For example, flap angle is recorded in degrees with floating-point precision by the UAS. Assume an acceptable reading is 5.045 degrees. In contrast, landing gear status is recorded as either up or down (Boolean: 1 or 0). Data typing enforcement ensures flap angle is stored in and retrieved from the DBMS as 5.045 rather than being rounded to an integer value of 5. Likewise, landing gear status is stored and retrieved appropriately as either a 0 or 1. These data types (float and Boolean) can be specified in schemas and enforced by some DBMSs. Users should consider the data type and precision requirements. If the user's elements have well-known types and precision and the user wishes to ensure they are maintained within the DBMS, then data typing enforcement is important. If not, it is less important.

The third aggregate evaluation criterion is *large aggregate transactions* which refers to the DBMS ability to store, retrieve, and update large aggregates quickly (within a few seconds). For

this criterion, a large aggregate is defined as being larger than 1 terabyte (TB) in size. Assume for a UAS example, the combined size of all the variables of interest collected in a sampling period is 1.5 TB. In other words, an aggregate would be 1.5 TB. In this situation, DBMS support for large aggregate transactions is important. If not, then this criterion is less important.

Conversely, the fourth aggregate evaluation criterion is *small aggregate transactions* which describes a DBMS's ability to store, retrieve, and update small aggregates quickly. A small aggregate is defined for this criterion as being smaller than 1 kilobyte (kB). In the UAS context, if the combined size of all the sampled variables is 800 bytes, then the resultant aggregate would be 800 bytes. In this situation, small aggregate transactions are important, because the resultant aggregate is considered small (<1kB). If the resultant aggregate was larger than 1 kB, then the small transaction performance would be less important. It should be noted that users may have the need for both large and small aggregate transactions. Additionally, some DBMS excel only at large scale, while others perform well at both.

The fifth evaluation criterion is *manipulation*. Manipulation refers to the DBMS's ability to update elements within stored aggregates independently from other aggregates and elements. This behavior may be desirable depending on what relationships exist between stored aggregates. For example, assume a UAS flight occurred and the log data is loaded into a DBMS. Later it is discovered that the timing signal was initially corrupt resulting in 10 aggregates containing incorrect timing information. Since subsequent timing information was accurate, it can be used to calculate the timing information for the corrupted periods. To update this information in the DBMS, the appropriate timing element within the affected aggregates must be changed. The manipulation property enables these update operations to occur independently without the DBMS automatically performing cascading update to all aggregates. This ability to independently perform

updates distinguishes manipulation from cross aggregate consistency. Users should consider whether updates to elements, such as timing, in existing aggregates are likely. If so, manipulation is important for the database solution to provide. If not, this ability is less important than others.

Finally, *plasticity* pertains to the DBMS's ability to add or remove elements within stored aggregates. For example, assume a UAS flight took place and the log data is loaded into a DBMS. Adequate GPS data was collected for the entire flight. Now the user wishes to use the existing GPS timing information to calculate the UTC time for each sample and store the result as a new element with the aggregate (LabSat, 2016). The plasticity property enables the DBMS to add a "Calculated UTC time" element to each aggregate. Similarly, assume a UAS flight occurred and the log data is loaded into a DBMS. Later it is determined that the *Engine Speed* element was included but contained corrupt data for this mission. The user wishes to remove this element from all of the aggregates for the flight. Plasticity enables the DBMS to remove elements from existing aggregates. To evaluate the need for plasticity, users should considerations whether the need exists for the DBMS to support adding or removing elements in existing aggregates. If the need exists, plasticity is important. Otherwise, it is not as important for the database to provide this ability.

Summary

This chapter explored the characteristics of log data which make it unique from other types of data. Specifically, log data is generated from a wide variety of electronic systems. It is also produced as a secondary function of its associated system. Log data is also inherently related to time as it is a "sequence of records." The structure of log data can vary widely in both organization and contents. Otherwise, log data is similar to other data. The discussion of log data was intended to provide a context for the UAS log data use case which motivated the overall research problem to develop a suitable log data storage and retrieval system.

A variety of database systems and evaluation criteria were also covered in this chapter. The most significant systems for this research will be the relational and modern NoSQL database types. Specifically, these NoSQL types include key-value, document, columnar, and graph models. Additionally, 12 criteria were developed to compare key aspects of these database systems. These five systems will be evaluated using criteria and following the methodology described in the next chapter.

III. Methodology

This section outlines the methods, processes, and tools that will be used to evaluate the five database types for small-scale storage and retrieval applications. Specifically, the five database types are the relational and four common NoSQL types: key-value, document, columnar, graph. An Unmanned Aircraft System (UAS) use case will provide the context and scope for this approach.

A two-part study will be performed beginning with a laboratory simulation of representative implementations for each database type. The UAS system will also guide the design and implementation of the databases. The second part will be a field study to collect inputs from UAS log data users. The results of the literature review and both parts of the study will be used as inputs to the Analytic Hierarchy Process. This process was developed by T.L. Saaty to support decisions involving multiple criteria. The AHP approach also supports evaluation for criteria for which there is not established rating scale (How to make a decision: The Analytic Hierarchy Process, 1990) (On the invalidity of fuzzifying numerical judgments in the Analytic Hierarchy Process., 2007). AHP will be the mechanism to exercise the 12 criteria, established in chapter II, to identify a database that is best suited for the UAS users' needs. To review, the 12 criteria are as follows: Cross-Aggregate Consistency (CAC), Data Type Enforcement (DTE), Large Aggregate Transactions (LAT), Manipulation (M), Plasticity (Pla), Preprocessing (Pre), Query Complexity (QC), Query Omniscience (QO), Result Timeliness (RT), Small Aggregate Transactions (SAT), Structural Malleability (SM), and Transparency (T). The use case, simulation study and the assessment process will be described in this chapter.

Database Simulation Study

This study will focus on a use case involving the Unmanned Aircraft System (UAS) log data. This use case will be used to guide the study parameters to compare the five representative implementations of common database types. Data from ten flights spanning a three-year period, and two aircraft types will be prepared for, stored in, and retrieved by the databases. Additionally, ten questions provided by the log data user community will be used to assess retrieval aspects of the databases.

For this particular dataset, the contents of the log data have changed over the system's lifetime. Changes were driven by both hardware and software updates to the aircraft and controller stations. Specifically, new variables were added, removed, renamed, shared names with other variables, and/or were moved within the log data. This issue makes for unique data storage and retrieval challenges as traditional data import techniques, such as matching a variable to a relational table attribute, will fail when the contents are modified. Furthermore, retrieval methods may need to be adjusted to account for changes in contents as well.

UAS Log Data Use Case

For this use case, we consider a commercially-produced UAS system composed of an Unmanned Aerial Vehicle (UAV), a controller station, and a Line-Of-Sight (LOS) antenna operating as shown in Figure 9. From the controller station, operators can control the system from start, taxiing, takeoff, during flight, through landing, and powering down. While the system is in operation, the controller sends commands to the UAV and receives data from the UAV, both which are recorded into log files. Additionally, other flight status, timing, location, and communication information is recorded during the flight (Headquarters, United States Air Force, 2014).

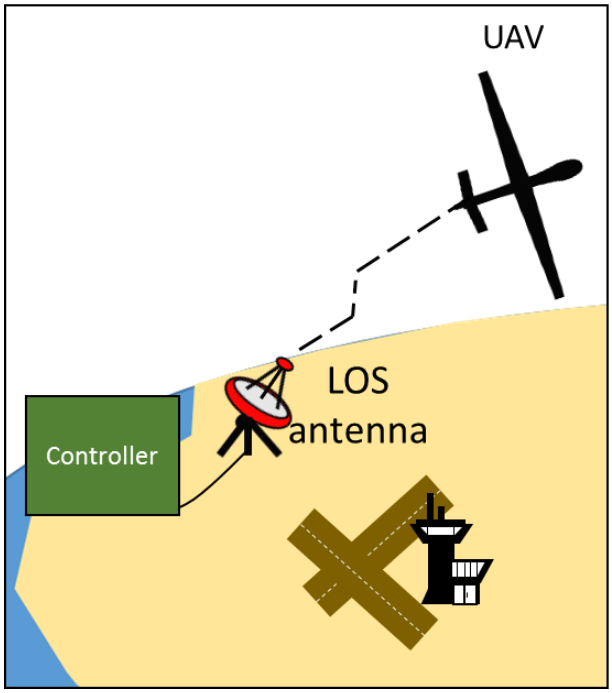

Figure 9. Operational View of the Unmanned Aircraft System of Interest.

Table 1 displays a list of log-generating subsystems and their descriptions. These subsystems will be used to organize data in the columnar and relational database types. In total, over 3,000 variables collect data from 15 subsystems during flight operations. Of these variables, approximately 800 are identified in documentation which is available to the system operators. However, the controller station records all variables and generates multiple log files throughout the flight. The system generates approximately 40 MB of log data per hour of operation.

Additionally, the log data is split into three categories of files, i.e., D, M, and A, and stored in a proprietary, but openly documented, MATLAB data format (Library of Congress, 2017) (Mathworks, 2018). The D-files contain variables whose contents alternate between Boolean values and are sampled at a low speed collection rate. In contrast, the M-files contain variables requiring double-precision accuracy and are also collected at a low rate. The A-files are a strict

subset of the variables in the M-files but are sampled at a higher collection rate. A set of D, M, and A log files are saved at the controller station for each hour of flight. The log data is not updated or changed after being written to these files.

Table 1. List of UAS Subsystems and Descriptions.

Subsystem	Description
Aircraft	Air vehicle system composed of structural, mechanical, and electrical subsystems and components.
Autopilot	Assists operator(s) with aircraft flight responses according to flight plan and environment.
Communication	Electrical and radio components enabling voice, data, and control communication between aircraft GCS, and other entities.
Electrical	Aircraft electrical components not directly supporting communications.
Engine	Aircraft component providing mechanical and electrical power to other air vehicle subsystems.
Failure	Responsible for tracking failure events in aircraft and GCS.
Feedback	Manages aircraft responses to environmental and operator input.
Fuel	Responsible for delivering fuel to aircraft engine and providing fuel status information.
Controller (Station)	Houses human operator(s) and ground-based command, control, communications equipment.
Landing Gear	Mechanical and electrical subsystem enabling ground mobility.
Modes	Manages navigation modes, waypoints, and planned flight profiles.
Navigation	Determines and tracks air vehicle location.
Operator Input	Responsible for accepting, tracking, and communicating operator input to appropriate UAS subsystem.
Performance	Tracks aircraft performance information such as air speed, angle of attack, and vertical velocity.
Sensor	Performs operations related to particular DoD UAS capability such as recording and transmitting video.

The differences in sampling rates resulted in modeling difficulties using a relational database. Evans refers to this type of problem as an *impedance mismatch* between objects and their relational models (Evans, 2011). Specifically, if data with both sampling rates are loaded into normalized tables, many null values must be stored as only a few variables are sampled at the higher rate. Thus, many tables and attributes will not contain data other than these null values. This is not an ideal situation for a relational database. The structural malleability criterion will be used to assess how the various database types handle this type of problem.

Another consideration is the need for specific data types, i.e., Boolean and double, to retain fidelity. The need to retain the appropriate precision will drive the need for the database types to support a data type specification. This capability will be assessed using the data typing enforcement criterion.

Table 2 presents a summary of the data used for the study. In this table, the *Flight Date* column contains the date the UAS executed a flight and generated log data. The *Aircraft Type* column indicates whether the UAS was type A or type B. The next column, Aircraft ID Number, contains a unique identification number for the UAV. Likewise, the Controller ID Number column contains an identification number for the controller system commanding the UAV. This information will be used to generate keys for some of the database types used in this study.

Table 2. Simulation Study Flight Data

Flight Date	Aircraft Type	Aircraft ID Number	Controller ID Number
2010-01-08	A	398	505
2010-01-12	B	433	505
2010-08-20	B	450	537
2010-12-14	A	371	537
2011-02-15	A	370	506
2012-01-06	B	433	522
2012-02-22	B	469	643
2012-04-17	B	450	643
2012-04-27	A	314	605
2012-05-29	A	336	605

Figure 10 contains the ten questions from the user community. These questions will be used to design queries that elicit the appropriate information from each database type.

1. For which flights, dates, and time periods does the type A aircraft engine exceed 2,500 revolutions per minute (RPMs)?
2. For which flights, dates, and time periods does the aircraft exceed 15,000 feet in altitude Above Ground Level (AGL)?
3. For which flights, dates, and times, did the airspeed exceed 100 Knots Indicated Air Speed (KIAS) with the landing gear extended in the down position?
4. Did any aircraft touch down in excess of 480 feet per minute?
5. What is the maximum altitude in feet AGL for each flight?
6. What was the operating time for each flight in seconds?
7. For which, if any, flights, dates, and times did the aircraft lose connection to the controller station?
8. For which, if any, flights, dates, and times, did the difference between the Manifold Air Pressure sensor #1 (MAP) 1 and MAP 2 exceed red/yellow limits for the type A aircraft?
9. For which, if any, flights dates, and times did the fuel pump fail during the mission?
10. For which, if any, flights, dates, and times did the Beta angle exceed 19 degrees during operations?

Figure 10. UAS Log Data Question List.

It is necessary to mention that no centralized enterprise-wide repository for storing and retrieving log data currently exists for the UAS of interest. Partial data stores exist on two separate networks, but neither instance contains a comprehensive data repository. The most accessible repository is maintained by the Flight Operational Quality Assurance (FOQA) analysts, but this data store contains only a portion of log data sets generally involving take-offs and landings. The other data store provides access, by exception, to complete flight data logs. Data is loaded into this repository when a data user requests analysis support from personnel located at another unit. Data is removed from this data storage medium when the original analyst deletes it. Separately from these data storage systems, analyst personnel typically obtain the log data from the controller station after a problem has occurred. Once their analysis is complete, the data is retained for a varying amount of time. That is, most analysts keep it on their location computer until they start running out of space.

This lack of a central repository for storing and retrieving log data, the *ad hoc* approach to sharing, and the single box approach to analysis makes this log data from this UAS appropriate for this study.

Database Design Simulation

The approach for designing and implementing each of the NoSQL database types follows a pattern involving six phases. In phase 1, the query expectations were considered in the context of each the data model, key-value, document, column, graph and relational. The considerations were used to define the appropriate aggregate construct to support log storage and retrieval via query. Once the aggregate was defined, phase 2 is initiated and the implementation-specific schema, i.e., logical data model, is designed. Queries are also developed at this time. For example, in the relational model, this step defines the tables, attributes, and keys. For the NoSQL types, this can include the definition of a schema or specification of the data type for elements within aggregates. Phase 3 involves preprocessing the data to be loaded into each database type. This phase includes the typical Extract, Transform, and Load (ETL) procedures related to preparing data to be loaded into a database. In phase 4, scripts are developed to create the appropriate schema for each database and import the data. Phase 5 involves importing the data using the scripts created in the previous phase. Lastly, the queries are run in phase 6. When designing and building the database simulations it was important to use identical processes for each simulation. It is understood there are numerous instantiations of NoSQL databases. For the purposes of this study, the base capabilities of each database type were of primary interest.

The single box environments were configured as virtual machines running a 64-bit Linux Mint 18.1 with 16 GB of RAM and 8 processing cores and 200 GB of hard disk space. This disk space

was allocated from a 2 TB Western Digital SATA drive rated at 7,200 RPM and 6 Gigabits per second. The virtual machines ran on an Intel i7-4820 CPU at 3.7 GHz with 32 GB of RAM on a 64-bit Windows 7 operating system. Each database type existed within its own virtual machine and only one machine was active at a time. This independence and isolation provided for uniform measurements between the database types.

A set of scripting tools written for the bash shell and in the Python language were developed to support these phases. The decomposition of the process into phases enable uniform measurements for phases 3 through 6 which can be compared during analysis. Specifically, bash scripting was used facilitate some data transformation, initialize the Python scripts for importing data, execute the queries, and record timing information.

Measurements

Table 3 identifies the characteristics to be measured in each phase of the study. Generally, two categories of data will be collected: the time to execute, i.e. the execution duration, and the lines of code (LOC) required to implement the steps in the phase. Specifically, the matrix columns identify the task areas of interest for the characteristic categories. The task areas of interest are *Create Schema*, *Transform Data*, *Import Data* and *Queries*. These task areas are the logical nomenclature for tasks accomplished in the phases as shown in this table. The cells containing an 'X' associate the phase with data collected. While phase 1 was important to the design, it produced no measurable quantity. However, phase 1 contributes to a qualitative assessment of the *Query Omniscience* criterion.

Table 3. Matrix of Measured Variables in Each Phase.

	Create Schema			Transform Data		Import Data						Queries	
	DTE	CAC	LOC	Execution Duration (s)	LOC	DTE	Execution Duration (s)	Size on Disk (MB)	# of Aggregates	LOC	T	Execution Duration (s)	LOC
Phase 2	X	X											X
Phase 3				X	X								
Phase 4			X							X			
Phase 5						X	X	X	X				
Phase 6											X	X	

Table 4 illustrates the relationships between the task areas and the evaluation criteria. Specifically, the *Create Schema* task contributes to the Cross-Aggregate Consistency (CAC), Data Typing Enforcement (DTE), and Query Omniscience (QO) criteria. The *Transform* and *Import Data* tasks both contribute to the Preprocessing (Pre) criterion. *Transform* and *Import Data* also contribute to assessing the Structural Malleability (SM) criterion. The *Import Data* task also contributes to the DTE, Large Aggregate Transactions (LAT), Small Aggregate Transactions (SAT), and SM criteria. The *Queries* task contributes to the LAT, Query Complexity (QC), and QO, Result Timeliness (RT), SAT, and Transparency (T) criteria. The analysis of the results will be discussed in detail in Chapter IV.

Table 4. Relationship of Task Areas to Evaluation Criteria.

	CAC	DTE	LAT	Pre	QC	QO	RT	SAT	SM	T
Create Schema	X	X				X				
Transform Data				X					X	
Import Data		X	X	X				X	X	
Queries			X		X	X	X	X		X

For this study, a single implementation of each database type was chosen as a representative example to be evaluated. The individual implementations were chosen from among popular

options providing features that best represented their type (Edlich, 2016). Specifically, Riak KV version 2.2.3 was selected to represent the key-value type, MongoDB version 3.6 was selected for the document class, HBase version 1.6 on Hadoop version 2.9 for columnar, Neo4j version 3.3.3 for graph, and MySQL Community Edition version 5.7 as the relational database representative. The author acknowledges each selected implementation still provides some nuanced characteristics and features extending beyond the representative traits of database type represented. Additionally, some databases may perform better in for certain tasks. These facts are unavoidable given that NoSQL implementations are often tailored to a particular purpose (Sadalage & Fowler, 2013) (Redmond & Wilson, 2012).

Analytic Hierarchy Process (AHP) Assessment

This section outlines how AHP will be used in this study to support the development of a single box storage and retrieval system suitable for log data. In 1990, Thomas L. Saaty developed AHP to evaluate multiple criteria and multiple alternatives towards a goal as a decision support mechanism. AHP involves organizing factors which contribute to the decision in a hierarchy which may be incomplete. The factors are evaluated in a pairwise manner by judges using a fundamental scale. The scale enables the judges to rate the importance for the pairwise comparisons. Additionally, the alternatives are compared against each other with respect to their performance for each factor. The ratings are recorded in a matrix. Using the results of the comparisons, a set of local priorities may be approximated by normalizing the values of the recorded judgements down the columns and then averaging across the rows, i.e., "the normalized column average." The result is a vector containing local priorities for the criteria and or factors. This process is repeated to determine local priorities for performance ratings as well. The last step is to calculate a set of

global priorities from the local priorities to identify the most suitable choice amongst the alternatives to support the goal (Saaty T. L., 1990).

Importance ratings

Table 5 presents the first matrix employed for this assessment. It consists of rows and columns containing the criteria. The full names of the criteria are provided in the rows and abbreviations are used as column headings. The relative importance of each criterion, with respect to the objective, is established using pairwise comparisons of criteria in each row to criteria in each column.

The comparisons are evaluated from left to right and then top to bottom in the matrix. Thus, Cross-Aggregate Consistency (CAC) is compared to itself, Data Typing Enforcement (DTE), Large Aggregate Transactions (LAT), and so on until CAC is compared to Transparency (T). Then DTE is compared to itself, LAT, and so on until DTE is compared to T. This process continues downward through the rows of the matrix.

Table 5. Log Data Storage and Retrieval System Evaluation Criteria Matrix.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1									X		
Data Typing Enforcement (DTE)		1										
Large Aggregate Transactions (LAT)			1									
Small Aggregate Transactions (SAT)				1								
Manipulation (M)					1							
Plasticity (Pla)						1						
Preprocessing (Pre)							1					
Query Complexity (QC)								1				
Query Omniscience (QO)									1			
Result Timeliness (RT)										1		
Structural Malleability (SM)											1	
Transparency (T)												1

Table 6 provides the fundamental scaled identified by Saaty for the the AHP process, with explanations tailored for clarity and purpose, to rate the pairwise comparisons. Acceptable values for the comparisons are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, and 1/9. If the criterion in the row is more important to the use case than the criterion in the column, a whole number is recorded in the appropriate cell. If the column is more important than the row, a fractional number is recorded (1990).

Table 6. The Fundamental Scale.

Importance intensity	Definition	Explanation
1	Equal importance	Two criteria, row a and column b, contribute equally to the objective.
3	Moderate importance of one over another	Experience and judgment moderately favor one criterion, row a, over another, column b.
5	Essential or strong importance	Experience and judgment strongly favor one criterion, row a, over another, column b.
7	Very strong importance	One criterion, row a, has demonstrated dominance in practice over another, column b.
9	Extreme importance	The evidence favoring one criterion, row a, over another, column b, is of the highest possible order.

2, 4, 6, 8	Intermediate values between the two adjacent ratings/judgements	Used when compromise is needed. For example, 6 can be used for the intermediate value between 5 and 7. Other intermediate values, such as 2.1, 2.2., 2.3, and etc., are acceptable if the whole numbers are deemed too substantial.
1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9	These values are used when you believe the criterion in the column, b, is more important than the criterion in the row, a. Note: Though fractions are easier human judges, decimal equivalent values are calculated used, in place of these fractions, for spreadsheet calculations.	

This process results in two notable outcomes. The first outcome is that each criterion is compared to itself along the diagonal. Thus, the diagonal of the matrix is filled with 1s. Additionally, a complete evaluation of the matrix would result in redundant comparisons. For example, the comparison of CAC to DTE has a reciprocal comparison of DTE to CAC. That is, the matrix is symmetrical about the diagonal. Thus, the comparisons to the right of the diagonal have a reciprocal comparison on the left side. Responses will be recorded to the right of the diagonal and the reciprocal will be calculated for the left. Therefore, half of the matrix is shaded in the figure to prevent redundant, wasted effort by the judges. Values will be calculated for the shaded area using the inputs from the right side of the diagonal.

Cells in the evaluation matrix are referenced in a row-oriented manner. For example, consider the cell in Table 5 containing the ‘X’ value. This cell represents the comparison between the CAC and RT criteria. To identify this cell in a shorthand manner, one may refer to this cell as the CAC-RT cell. Its symmetric pair (about the diagonal) would be the RT-CAC cell. The value in the symmetric pair is the reciprocal of the value in the original cell, CAC-RT. Thus, if CAC-RT contained a “2.00,” RT-CAC would contain “0.50.”

Inputs from subject matter experts (SMEs) from operations and maintenance support, training, safety, and Flight Operations Quality Assurance (FOQA) expertise were requested. These areas encompass the majority of the known population of active duty UAS log data users. The testing

community is not directly represented, but it was known that one of the operations and maintenance support SMEs provided support to this community.

A research goal was to obtain responses from at least two judges in each area to provide some validation of the methodology.

Performance Ratings

In addition to the judge's importance inputs, ratings must be assessed for the database types with respect to each criterion. In these cases, pairwise comparisons are made between the database types regarding how well each type provides capability, i.e. performs, as defined by a criterion under consideration. Twelve additional matrices are used to record assessments made using data from the comparison study and database type's fundamental capability, i.e. baseline ability, to provide features as identified by the literature review. The same scale is used for these ratings, but instead of importance, performance is considered. As before, these matrices will be used to calculated additional local priorities as described by Saaty (1990).

Table 7 presents an example matrix that will be used to record these ratings. In this matrix, both the columns and rows contain a database type. The cells in the matrix will be populated with the ratings as described previously. The diagonal of this matrix again contains "1's," signifying that each database type is rated equally to itself with respect to the considered criterion. Furthermore, these matrices are also symmetric about the diagonal and thus the cells to the left of the diagonal are shaded as before. The shaded area is shown to illustrate the symmetry of the matrix; values will be recorded in this area of each matrix.

Table 7. Example Matrix for Performance Ratings.

	Key-Value	Document	Columnar	Graph	Relational
Key-Value	1				
Document		1			
Columnar			1		
Graph				1	
Relational					1

Global priorities

Combining the results of both sets of ratings will be used to calculate the global priorities. The global ratings will be interpreted to identify the most suitable database type for each SME's inputs. That is, each SME will have a database type identified as most suitable given their inputs. The database type with the valued global priority is considered the most suitable. The results of this methodology are calculated and discussed in Chapter IV.

IV. Results

This chapter presents the results of the simulation study, field study, and the AHP assessment to support a decision involving which database is most appropriate to store and retrieve UAS log data in a single box environment. The simulation study data is discussed first to lay the foundation for the AHP assessment's calculations. The assessment will use a combination of the literature review findings and the simulation study's results to develop local priorities related to the performance of the database types. Once the performance ratings are calculated, the importance ratings will be presented and analyzed. Finally, the global priorities will be calculated using both the performance and importance ratings to determine the most suitable database for the UAS log data use case.

Simulation Study and Performance Ratings

The results from the database simulation study are discussed in this section. A definition of the aggregate for each database type is presented first. Next, these results are organized and presented according to the task areas of interest identified in Chapter III, Table 3 to provide clarity. To review, these task areas are *Create Schema, Transform Data, Import Data* and *Queries.* Afterwards, each the results from each task area will be used to support the performance assessment of each database type. Chapter III, Table 4 illustrated the relationship between task areas and criteria.

The performance evaluation matrices were filled according to the AHP rating process and scale described in Chapter III. Since the matrices are symmetric, the cells to the right of the diagonal contain recorded values and those to the left of the diagonal contain calculated reciprocals of the corresponding rating. Therefore, the ratings recorded in the "right" cells will be the focus of

analysis and discussion. Subsequently, the calculated ratings in the "left" cells will not be comprehensively discussed.

Aggregate Design

A quick review of terms is provided here for clarity. For this discussion, an *aggregate* is composed of one or more data elements and is considered an atomic unit to the DB during transactions. An *element* contains a value and is similar to a cell in a relational database table. A *transaction* is a CRUD operation performed on a single aggregate by the DB. Lastly a *query* is a set of transactions performed by the DB on one or more aggregates and elements.

For this study, aggregates were tailored for each database type to exploit the capabilities of the underlying data model. In general terms, an aggregate contained a uniquely defining characteristic and a set of related elements. Though the definition of an aggregate in a relational database context has a different interpretation, the equivalent comparison for this discussion would be a row in a table or in some cases, the table itself. Multiple aggregates, on the order of hundreds of thousands, were stored in each database type.

Riak KV Aggregates

In the KV model, an aggregate consisted of the value portion of the key-value pair. Unique keys were defined using a concatenation of the flight metadata presented in Chapter III, Table 2, the flight number of the day, and the value of the running count of samples beginning at 0. The value portion of each stored key-value pair consisted a set of key-value pairs containing the variable name and value for all defined UAS log data variables collected for a given sample.

Figure 11 and figure 12 present an example of a key and the resultant aggregate respectively. As shown in figure 11, the key includes an "A" prepended to the aircraft ID to indicate the data

was generated from a type “A” aircraft. Likewise, a “C” was prepended to the controller ID. This addition makes the data more human-readable and conform to the Riak key requirement enforcing that all keys are strings (Basho Technologies, Inc, 2018). Elements stored within the aggregate are not individually accessible via database queries; the entire aggregate must be retrieved and subsequently parsed outside of the database’s native functions.

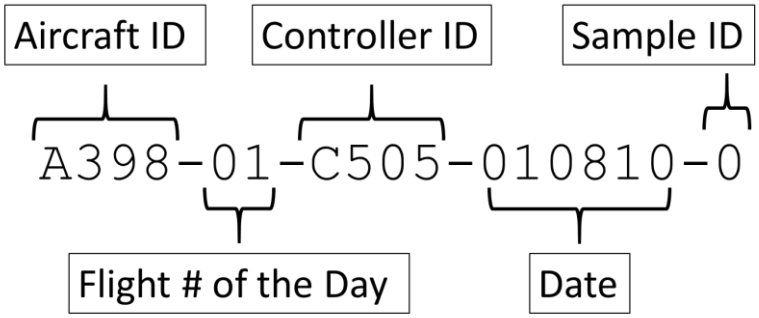

Figure 11. Example of a Unique Key Using Flight Metadata and Sample Id.

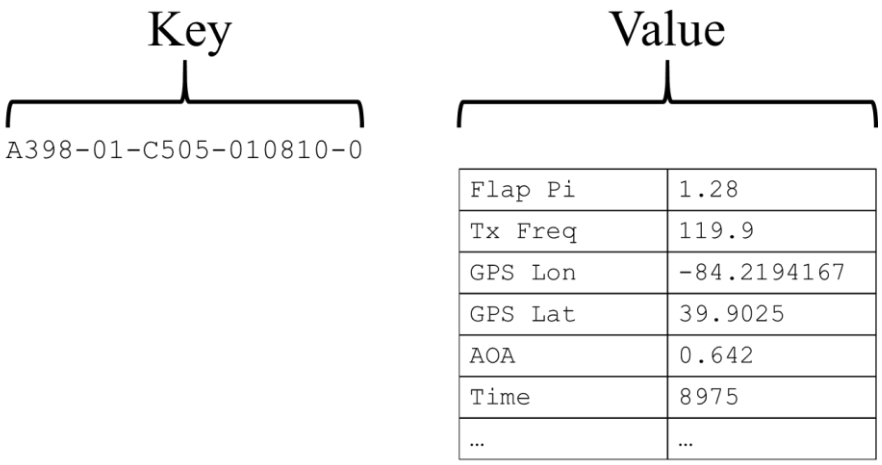

Figure 12. Example Riak KV Aggregate.

MongoDB Aggregates

The aggregate construct for the Mongo document database was similar to the aggregate model used for the Riak KV database. As discussed in Chapter II, document databases are composed of collections and documents. In this study, the collections were designed to contain a single flight's log data set. Collections were named using the aircraft tail ID, the flight number of the day, the controller ID, and the date in the same manner described for the Riak KV keys. The resulting collection names are similar to the key shown in figure 11, yet lacked the sample ID number.

Figure 13 provides an example of an aggregate organized using the MongoDB document model. In the document data model, the documents are the aggregates. In MongoDB, the documents are identified by a document_*id* element that contains a value that is unique to the collection's namespace (Chodorow, 2013). These document IDs can be automatically generated or manually assigned. For this study, document IDs were manually assigned to each aggregate using the same string constructs as shown in figure 11. The remaining elements of the document consisted of key value pairs, consisting of log data variables name and values. Each element contained in the document is accessible to the user.

_id	A398-01-C505-010810-0
Flap Pi	1.28
Tx Freq	119.9
GPS Lon	-84.2194167
GPS Lat	39.9025
AOA	0.642
Time	8975
…	…

Figure 13. Example MongoDB Document Containing UAS Log Data.

HBase Aggregates

For the HBase database, aggregates were designed to store and retrieve the data required to support the queries identified in Chapter III. For each query, a table was created to store the data that the query would need to retrieve. Within each table, a column family was created to group together the log data for each flight. An aggregate is a row from a table and its elements vary to provide the information needed by a particular query. Additionally, a *Flights* table was created with column families named after the UAS subsystems to store all of the data collected. This design enables HBase to store all of the available log data, support the queries known at the time of design, and is consistent with the NoSQL paradigm of organizing data as it exists and how it is intended to be used (Sadalage & Fowler, 2013) (Redmond & Wilson, 2012) (Serra, Choosing technologies for a big data solution in the cloud, 2017).

Figure 14 displays the relationships between elements of the *Query #1 (Q1)* table in HBase. The column families organize the log data together for each flight. Within the column families, only two columns are stored: *Engine Speed* and the *Sample Date/Time*. The *Engine Speed* column contains the values sampled and the *Sample Date/Time* column contains the date and time the sample was collected. The remaining nine *Query* tables are designed in a similar manner to store only the data that will be retrieved when the corresponding query is executed.

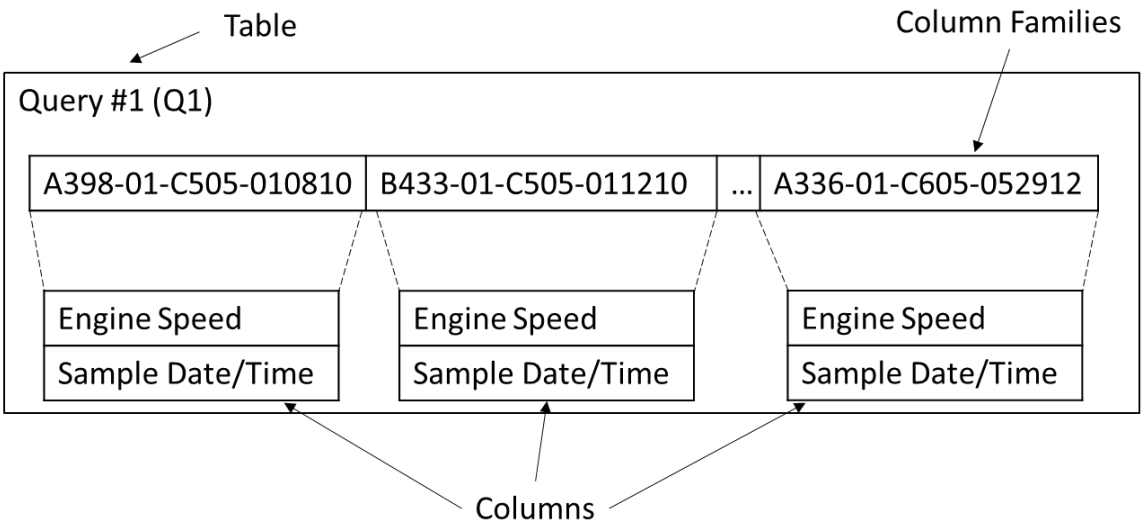

Figure 14. HBase Query #1 Table Logical Overview.

Figure 15 presents the logical relationships between the column families and columns in the *Flights* table. In this diagram, only six of the 15 column families are shown. The complete list of column families includes *Aircraft, Autopilot, Communications, Electrical, Engine, Failure, Feedback, Fuel, Controller Station, Landing Gear, Modes, Navigation, Operator Input, Performance,* and *Sensor*. Additionally, only seven of the hundreds of columns are shown for two of the column families. *Power, Rx Frequency, Sample Date/Time, Tx Frequency, Speed, and Temperature* are included in this diagram.

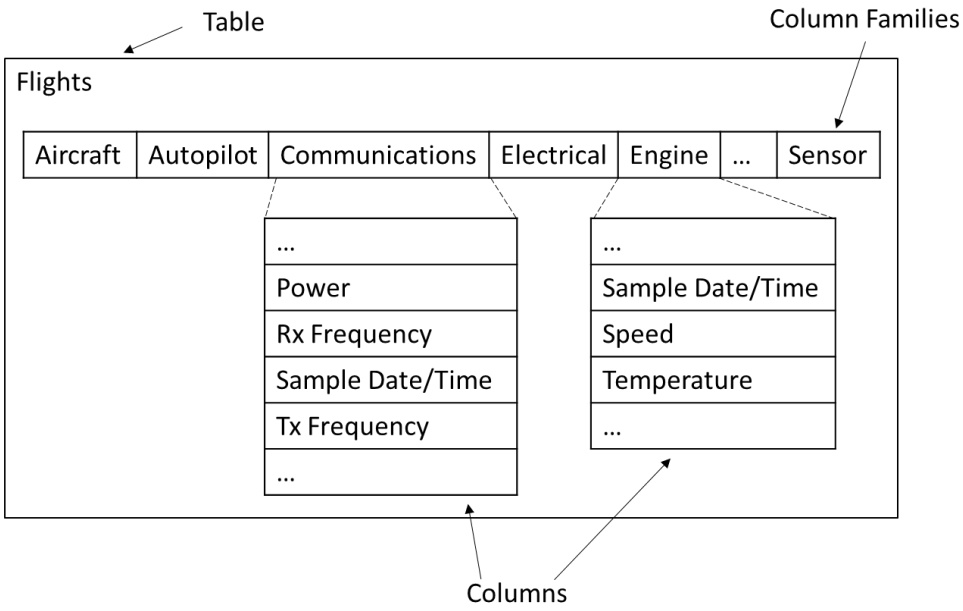

Figure 15. HBase Flights Table Depicting Column Families and Columns.

The reader should observe that *Sample Date/Time* is a repeated column in the *Query* and *Flight* tables across multiple column families. This redundancy is a characteristic of NoSQL designs. It intentionally violates the traditional relational model's principle of normalization. In turn, this violation provides performance gains obtained by storing data together, i.e. the principle of locality (Denning, 2005).

Neo4j Aggregates

Recall that Neo4j refers to graph vertices as *nodes* and data stored within edges or nodes as properties (Robinson, Webber, & Eifrem, 2015). Additionally, edges represent relationships between nodes. Though they are shown with arrows to indicate direction, the edges/relationships are treated as bidirectionally traceable by Neo4j. That is, queries can trace the relationship along an edge from either node. Since both nodes and edges can store properties, aggregates in the Neo4j database can be either a node- or an edge-type depending on the details of a transaction. Thus,

properties stored in the nodes and edges are elements. This terminology will be used interchangeably within this document.

Figure 16 displays the schema used to organize log data in the Neo4j database. In this diagram, *Tail ID, Flight, Controller ID,* and *Sample* are the node-type aggregates. *Tail ID* aggregates, i.e., nodes, contain data elements about the aircraft such as its ID number, aircraft type, its date of entry into service, date of last overhaul, assigned operating location, or other aircraft specific information. Similarly, *Controller ID* aggregates, i.e., nodes can store ID number, location, or other controller specific information. *Flight* aggregates, i.e. nodes, store data elements about the starting and date and time of the flight, the software versions of the aircraft and controller and other common/related data between aircraft, controllers, and samples. The *Sample* aggregates, i.e., nodes, store the bulk of the log data generated for a flight. Hundreds of elements, corresponding to sampled variables, are contained within each *Sample*. Thousands of *Sample* aggregates, each corresponding to an individual sample in time, are associated with a particular *Flight* aggregate.

Flew, *Controlled*, *Collected For* are the edge-type aggregates. In this implementation, *Flew* and *Controlled* aggregates contain no log data elements. These aggregates simply represent the relationships between *Flight* and *Tail ID* and *Flight* and *Controller ID* respectively. The *Collected For* aggregate contains one element storing the date and time the sample was collected.

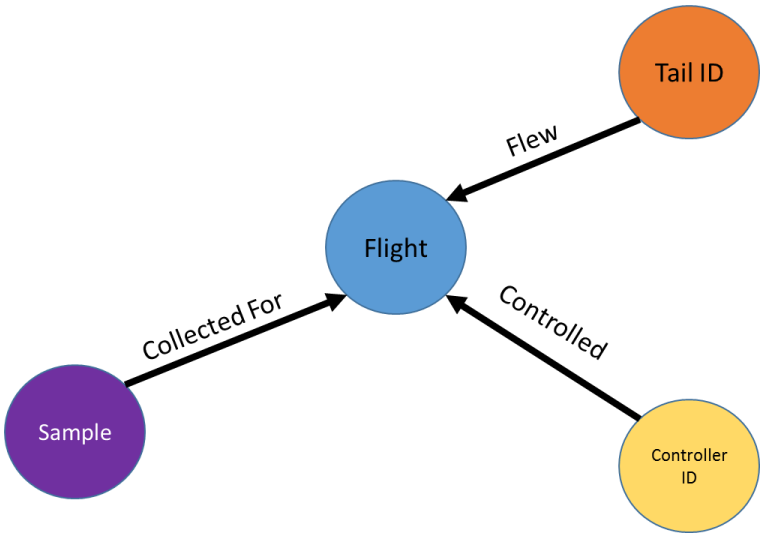

Figure 16. Neo4j Schema for Log Data.

MySQL "Aggregates"

In this methodology, the use and meaning of the term aggregate in the context of a relational database differs from the traditional definition. Here, a relational aggregate shall be interpreted as a row in a logical table that is treated as an atomic unit. A logical table includes tables defined in the relational database's schema as well as views that are subsequently created. This definition enables comparison of aggregate types, related behaviors, and capabilities between the NoSQL and relational database types on more even terms. These comparisons will be explored more thoroughly later in this chapter.

Figure 17 displays an example of the *Engine* subsystem table in MySQL. Part (a) provides a portion of the schema definition. Specifically, the attributes and primary key designations are shown. This table and other subsystem tables are normalized to approximately the third normal form (Elmasri & Navathe, 2016) (Hoffer, Venkataraman, & Topi, 2016). Part (b) illustrates example rows in the table with values shown for the composite primary key. In this example, the

rows are considered aggregates and the attributes are the elements. Other tables follow the same pattern concerning the relationship between aggregates and rows and elements and attributes.

Engine	Attribute	Key
	flight_id	Pri
	sample_id	Pri
	speed	
	temperature	
	…	

(a) Partial Schema Definition of Engine Table

Engine	flight_id	sample_id	speed	temperature	…
	1	0	…	…	…
	1	1	…	…	…

(b) Example Engine Table Data

Figure 17. MySQL Engine Table Example

Aggregate Summary

These sections outlined aggregate and element definitions and provided specific examples of each for the database types involved in this study. The key-value and document aggregates are similar, but the KV database does not provide direct, independent access to the individual elements. The columnar aggregates are essentially rows within this data model. The graph database provides two type of aggregates, i.e. nodes and edges, in which elements can be stored and retrieved. Lastly the operating definition of an aggregate for the relational database is a table's row. These concepts provide the foundation for the subsequent discussions concerning database comparisons.

Task Area Results

Create Schema

The *Create Schema* task area includes all activities related to preparing the database itself to import log data. For MySQL, this includes defining tables, attributes, data types, constraints, keys, some indexes, and various other options. For the Riak database, this included identifying the expected data types and indexes for values on which "advanced" queries would be executed. An

"advanced" Riak query is considered any query that is not exclusively a value look-up using a specified key. In MongoDB, data types and indexes can be defined to speed up any expected queries. Additionally, collections can be defined to further organize the data and exploit the principle of locality. For HBase, tables and column families must be defined to organize the data. Finally, Neo4j used a schema definition to define nodes and any required relationships.

Table 8 displays the results for the Create Schema task area of this study. In this table, the rows contain results for each of the five database types. Additionally, the specific database implementation is identified in the row label. In the *DTE* column, a categorical value of *Yes* or *No* was recorded. *Yes* indicates that a data type definition could be specified and subsequently enforced for aggregate elements. *No* indicates the opposite; that a data type could not be specified and/or enforced by the database itself. Likewise, the *CAC* column also contains categorical values. Results, *Yes* or *No*, in this column indicate whether the database natively permits, tracks, and enforces relationships or constraints between aggregates. The term, *Natively*, implies that additional third-party software is not required to facilitate this capability. The value included in the *LOC* column indicates the total number of Lines of Code (LOC) required to support all activities related to this task. The LOC value only includes code written and/or generated during the study. Generated LOC are those which were produced by other code written in this study. For example, code written in the Python language was used to generate JavaScript and Cypher Query Language code.

Table 8. Summary of Create Schema Results.

	Data Typing Enforcement (DTE)	Cross-Aggregate Consistency (CAC)	LOC
KV (Riak)	Yes	No	147
Doc (Mongo)	Yes	No	657
Col (HBase)	No	No	523
Graph (Neo4j)	No*	Yes	34
Rel (MySQL)	Yes	Yes	1536

Three observations were made from these results to assess the performance of the database types. Recall from Chapter III Table 4, this task is related to CAC, DTE, and Query Omniscience (QO) criteria. The observations will be discussed and the performance ratings for CAC will follow. Since DTE and QO are dependent on the *Import Data* and *Queries* tasks respectively, their ratings will be withheld until their associated tasks are discussed.

DTE Observation

The first observation was the column and graph database types do not support specifying a data type during schema creation. Specifically, HBase stores all data as byte arrays and provides no enforcement of data type. In contrast, Neo4j does not support data type specification in schema definitions, however it does support specifying a data typing during importing and loading. Riak KV supported string, double, and Boolean data types which are the types used in the UAS of interest's log data. MongoDB and MySQL support specifying these types among others. In fact, MongoDB's relatively high number of LOC is related to its ability to specify data types. Furthermore, relational database type requires a comprehensive specification of the intended data type at the time of schema design. Ultimately, the columnar database type will be rated lower compared to its peers for the DTE criterion. The ratings for DTE will be discussed in the *Import Data* section of this chapter.

CAC Observation

A second observation from the *Create Schema* task area was related to NoSQL's structuring of data and the resulting impact to the query execution. If the data is not structured in such a way that a query can access it, then the database may have a limited ability to perform retrieval queries. For example, the design of the HBase column database's structure, i.e. schema, affects how easily a query can obtain the data. Specifically, if the data is not stored in the same column family, multiple queries must be run with their results combined. HBase does not provide a native mechanism to combine query results. Thus, third party software must be used to perform these merging operations. Likewise, for KV databases, filtering pertinent data out of query results requires assistance from third party software if the storage structure does not exactly match the retrieval query. The remaining database types did not demonstrate significant query limitations due to schema design.

This observation influences a lower rating for the KV and column database types compared to the other database types for the Query Omniscience (QO) criterion. A lower rating is necessary because the baseline capabilities of these database types do not readily support queries that are developed after the database is designed. Specifically, the developer should understand that queries must be well-known at the time of design and that additional queries may be difficult to implement after data is loaded. The specific ratings for QO will be discussed in the *Queries* section of this chapter.

The last observation was that only graph and relational database types provide CAC support. The relational database, again, requires a complete definition of relationships and constraints during schema design. However, the graph database also requires complete specification of these properties between aggregates. Both database types store and perform operations on the data

according to these defined characteristics. The large number of LOC for the relational database hints at this fact. In contrast, the small number of LOC for the graph type does not suggest the importance of these definitions. The nature of the edges is defined by the model and once the relationship is defined, the graph database engine understands how to handle them during transactions and queries (Robinson, Webber, & Eifrem, 2015). The ratings for graph and relational database types will be significantly higher than those for KV, document, or column types.

CAC Ratings

Table 9 displays the CAC criterion ratings for the database types in this study. The row and column titles are generalized to the database type to illustrate the ratings reflect both the literature review and the results from the study. That is, the names, Riak KV, MongoDB, HBase, Neo4j and MySQL have been removed from these headings. As discussed in Chapter III, cells in this matrix contain values which represent comparisons between the database type identified by the row heading and the type identified in the column. In this matrix, the cells along the diagonal contain the value "1.00" indicating equal performance because these cells represent self-comparisons. In other words, a comparison of the KV database's DTE performance to itself is equal. Furthermore, only the ratings to the right of the diagonal will be discussed because the matrix is symmetrical about the diagonal. Thus, values to the left of the diagonal are reciprocals of values to the diagonal's right. These facts about the diagonal will be true for all AHP matrices.

The first data row of this matrix contains the results of the pairwise comparisons for the KV database type. None of the KV, document, or column types natively support consistency between aggregates nor do they automatically store, retrieve, or update relationships. Thus, the KV-Doc and KV-Col cells contain a 1.00 to indicate their CAC performance is equal. Their symmetric pairs, i.e., Doc-KV and Col-KV cells, contain the reciprocal value of 1/1=1.00. In sharp contrast,

the graph and relational types were extremely proficient at managing consistency and relationships. Therefore, KV-Graph and KV-Rel cells contain 0.11 which indicates that graph and relational types are exceptionally better than key-value types for this criterion. Their symmetric pairs, i.e., Graph-KV and Rel-KV cells, contain reciprocal values, i.e. 1/(0.11)=9.00. The document and column type rows contain values reflecting that column and document type's performance was equally poor and that the graph and relational types performed exceptionally well compared to these types. Graph and relational types are were assessed as equal for this criterion. The symmetric pairs to the left of the diagonal contain the reciprocal values for these ratings.

Table 9. CAC Performance Rating Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	1.00	1.00	0.11	0.11
Doc	1.00	1.00	1.00	0.11	0.11
Col	1.00	1.00	1.00	0.11	0.11
Graph	9.00	9.00	9.00	1.00	1.00
Rel	9.00	9.00	9.00	1.00	1.00

Transform Data

The *Transform Data* task area includes activities related to preparing the data to be imported into the database. In general terms, these activities are consistent with the *Extract* and *Transform* phases of traditional ETL (Elmasri & Navathe, 2016) (John & Misra, 2017) (SAS, 2017). For this study, it includes transforming the data from the proprietary MATLAB format into a format readable by each database type. Specifically, the Comma-Separated Value, (CSV) format was selected because all database types included mechanisms to import data in this format. Additionally, data validation checks were performed in this task area to ensure that the UAS log data was complete and did not contain errors. The transformation tasks are considered by the Preprocessing and SM criteria.

Table 10 provides a summary of the results from the *Transform Data* task area of this study. In this table, each row of data contains results for one of the database types. The *Execution Duration* column indicates the cumulative number of seconds lapsed while transforming data from all 10 flights. Values in the LOC column display the total number of lines of code written and/or generated, during the study, to execute activities associated with this task area.

Table 10. Summary of Transform Data Results.

	Execution Duration (seconds)	LOC
KV (Riak)	2154.6	427
Doc (Mongo)	2014.7	427
Col (HBase)	2331.6	885
Graph (Neo4j)	1695.5	321
Rel (MySQL)	616.1	1378

Two observations were made from this portion of the study and applied to the assessment of the database types for the relevant criteria. Recall from Chapter III, table 4 that *Preprocessing* and *Structural Malleability* (SM) was associated with the *Transform Data* task area. Additionally, both criteria were also dependent on the *Import Data* task areas. Therefore, the observations will be described, but the discussion of ratings will be postponed until the all associated task areas have been reviewed.

SM and Preprocessing Observations

The first observation was that despite claims that NoSQL databases ingest "raw" data, there is still much work involved to transform the data into something which can be imported (Sadalage & Fowler, 2013) (Sullivan, 2015) (Robinson, Webber, & Eifrem, 2015). The fact that the LOC is non-zero for this task highlights this fact. However, it should also be noted that the required number of LOC is smaller for the NoSQL types than it is for the relational type. The relational database

still required more work to transform and prepare the data. This included removing special characters, white space, and often adding quotations to variable names in the raw data. Additionally, the sampled variables needed to conform to expectations for a matching process which would pair them with attributes defined in the relational database's strict schema. This was not necessary for the NoSQL types which were accommodating to deviations from schema expectations.

The tradeoff for this flexibility is encountered during query design and execution. Queries must account for differences in the structure of aggregates, i.e. different aggregate types. For example, if a sampled variable is named *Eng Spd* in one data set, *Engine Spd* in another, and *Engine Speed* in a third, the NoSQL types are flexible enough to allow these three aggregate variants to be permitted when log data is imported despite any predefined schema expectations. In contrast, these variants must be matched to a particular table and attribute for the relational database. The SM criterion considers these types of issues. As noted, ratings for this criterion will be discussed in the *Import Data* section of this chapter.

Another observation was that NoSQL types required more execution time to transform the log data into the CSV format. The difference between the NoSQL and the relational transformation processes results from the way the A-files were prepared and a design choice in the relational model to handle data sampled at a different rate. Since the A-files consist of a strict subset of M-file variables at a higher rate, storing these rows, i.e. "aggregates" in normalized tables would result in hundreds of attributes, i.e. "elements" containing null values. In fact, more A-file aggregates types would exist for a log data set than the others combined due to the high sampling rating. The end result would be that some RDB tables would contain hundreds of null attributes

for hundreds of thousands of rows. To avoid this result, a separate A-file table was created to store this data in the relational database.

This A-file issue was not a problem for any of the NoSQL types. Sparsely populated tables and different "types" of aggregates are some of the reasons NoSQL solutions were developed. Thus, instead of separating the A-file log data for the NoSQL types, this data was merged with the low-speed D- and M-file data. Merging this data was an intensive process to match and verify the merge was performed correctly. Hence, the execution times for the NoSQL databases were all longer than the transformation execution time for the relational database.

Import Data

The Import Data task area included activities involving importing the transformed data into each database type. These activities are generally consistent those of the *Load* phase of traditional ETL (Elmasri & Navathe, 2016) (John & Misra, 2017) (SAS, 2017). Specifically, the activities involved ensuring the data was located at the appropriate file system location for which each database type expects to import data, performing matching between aggregates and schema definitions, and other operations to load and store data within the database types.

Table 11 presents a summary of the results from the Import Data task area of this study. The rows of data contain the results for the chosen implementation of each database type. . In the *DTE* column, a categorical value of *Yes* or *No* was recorded. *Yes* indicates that a data type definition could be specified during this phase of the study. *No* indicates the opposite; that a data type could not be specified. Like other tables, *Execution Duration* column displays the cumulative number of seconds lapsed while the Import Data activities operated on the 10 flights of log data. The *Size on Disk* column provides a measurement in megabytes (MB) for the total size of the data once it was loaded into each database type. The *# of Aggregates* column identifies the total number of

aggregates stored in each database. Lastly, the *LOC* column contains values indicating the total number of Lines of Code written and/or generated, during the study, to facilitate activities associated with this task area.

Table 11. Summary of Import Data Results.

	DTE	Execution Duration (seconds)	Size on Disk (MB)	# of Aggregates	LOC
KV (Riak)	No	11,815.9	38,844	4,992,021	119
Doc (Mongo)	No	7405.1	2,293	4,992,021	29
Col (HBase)	No	8,722.1	33,213	19,123,233	50
Graph (Neo4j)	Yes	7,950.0	19,455	9,984,212	10,624
Rel (MySQL)	No	763.0	1,899	9,237,123	913

A few observations were made during this segment of the study and combined to assess the database types using the evaluation criteria. Recall from Chapter III, Table 4 that Data Typing *Enforcement (DTE), Preprocessing (Pre), Structural Malleability (SM), Large Aggregate Transactions (LAT), and Small Aggregate Transactions (SAT)* criteria are associated with this task area. The ratings for DTE, Pre, and SM will be discussed in this section. However, the LAT and SAT criteria are also dependent on the *Queries* task area. Therefore, their ratings will be discussed in the *Queries* section to ensure they have been comprehensively explored.

SM Observation

One observation is the number of lines of code required for the graph database type. This relatively large number results primarily from the code generated to load various aggregate "types" into the database. As discussed in the Transform Data section, when a variable name in the raw data changes, the NoSQL types can adapt to store the data despite any existing schema expectations. For the graph database type, each name change can be handled, but the import process must specify the how to do so.

Figure 18 presents three examples of how the Neo4j Cypher code can be modified to import data from a variable with a changing name. Specifically, this graph database can store *Eng Spd, Engine Spd,* and *Engine Speed* as long as the specific aggregate element, i.e. *sample.eng_spd, sample.engine_spd,* or *sample.engine_speed,* into which the variable will be stored is explicitly named in the import code. In a sense, this method matches the variable name to an aggregate element that is specified at the time of import. Fortunately, the import code can be generated from the transformed data using a programming language such as Python.

```
sample.eng_spd=toFloat(data.`Eng Spd`)

sample.engine_spd=toFloat(data.`Engine Spd`)

sample.engine_speed=toFloat(data.`Engine Speed`)
```

Figure 18. Three Examples of Cypher Code to Import a Variable with a Changing Name.

This capability supports SM in which various aggregate types can be loaded into the database. The other NoSQL database types generally do not require such explicit definitions and can infer more from the data itself. Thus, Neo4j offered the same capability, but it required more work.

SM Ratings

Table 12 displays the SM criterion ratings matrix for the studied database types. The KV and document databases equally support the addition of new aggregate types, so the KV-Doc cell contains 1.00 to indicate this equality. In both cases, new aggregates can be loaded without restarting the databases or making significant changes. For columnar databases, new aggregates can require new column families to be defined. Adding new column families to a columnar database typically require a database restart (Sadalage & Fowler, 2013) (Sullivan, 2015) (Redmond & Wilson, 2012). Thus a 5.00 was recorded in KV-Col and Doc-Col cells to indicate a

stronger performance by the KV and document types that the columnar type for this criterion. Since the graph database required additional code to support additional aggregate types, its performance was not rated as strongly as the KV and document types. Hence the KV-Graph and Doc-Graph cells contain a value of 2.00 to show slightly better performance by the KV and document types.

Table 12. SM Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	1.00	5.00	2.00	9.00
Doc	1.00	1.00	5.00	2.00	9.00
Col	0.20	0.20	1.00	0.14	5.00
Graph	0.50	0.50	7.00	1.00	9.00
Rel	0.11	0.11	0.20	0.11	1.00

The relational database type was least suited to accept additional aggregate types. Recall that a new aggregate type containing a new element in this context would require one or more tables to be modified to support a new attribute. This process involved updating the schema, restarting the database, and depending on the implementation and design will produce tables full of default values in the new attribute columns for the updated tables. The default values may be valid and acceptable or may be required to be set to a sentinel value, etc. In either case, this will require additional work to check, set, and verify these elements for previously existing "aggregates." Thus, all the NoSQL types were given better performance ratings than the relational type for this criterion. Specifically, the column type was rated as performing strongly better, i.e., rated 5.00, and the other NoSQL types were extremely better, i.e. rated 9.00, than relational.

DTE Observation

Taking another look at figure 18, the reader should also note the method to specify the data type is shown. Specifically, *toFloat( )* is used to convert the data and bind the aggregate element to a float type with floating point precision. Other data types are available as well (Neo4j, Inc., 2018). This observation was combined with the DTE discussion from the Create Schema section and the Chapter II literature review findings to determine DTE ratings for each database type.

DTE Ratings

Table 13 presents the DTE criterion ratings for the five database types evaluated. The first data row of this matrix contains the results of the pairwise comparisons for the KV database type. Since the KV database supported some data typing but did not support as many types as the document database. Thus, a value of 0.33, was recorded in the KV-Doc cell to indicate the document database type performs DTE moderately better than KV types. However, the KV type performs DTE moderately better than the columnar type, so a value of 3.00 was recorded in the KV-Col cell. In contrast, the graph types enable strong control over data types and therefore a 0.20 was recorded in the KV-Graph cell. Likewise, the relational database type provides the strongest possible control of typing, so the KV-Rel cell contains 0.11 indicating this fact.

Table 13. DTE Performance Rating Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.33	3.00	0.20	0.11
Doc	3.00	1.00	7.00	0.20	0.11
Col	0.33	0.14	1.00	0.13	0.11
Graph	5.00	5.00	8.00	1.00	0.20
Rel	9.00	9.00	9.00	5.00	1.00

Document type databases performed DTE significantly better than column databases, but not as well as graph or relational types. To represent these observations, Doc-Col was rated 7.00, Doc-Graph was 0.20, and Doc-Rel was 0.11. The difference between Doc-Graph and Doc-Rel suggests that relational is still the best of the studied types. Likewise, the Col-Graph and Col-Rel cells provide consistent ratings of 0.13 and 0.11 respectively. Lastly the Graph-Rel cell contains a 0.25 rating suggesting that relational types are still better than graph types for DTE because relational types still have a moderate advantage.

Preprocessing Observations

Another observation concerning data import is the amount of time required to import the data and the size on disk. If the *Execution Duration* results are viewed independently, they do not appear to suggest any coherent results. For example, the NoSQL types total import time ranges from 405.1 to 11,815.9 seconds. Additionally, the relational database's import time falls within the range for the NoSQL types at 763.0 seconds. However, when the *Execution Duration* is considered in conjunction with the *Size on Disk*, a story materializes. Even without employing statistical methods, there appears to be a correlating increase in import duration with an increase in the amount of data stored in the database on disk.

It should come with little surprise that the data normalized in the relational database requires the least disk space of the databases studied. This fact is a benefit of normalizing data in a relational database to mitigate the need for redundantly stored data (Codd, 1970) (Elmasri & Navathe, 2016) (Redmond & Wilson, 2012). Likewise, because the NoSQL databases often intentionally store redundant data, their size on disk is categorically larger than the relational databases. However, what was unexpected was the amount of time required to support this redundancy—specifically for the KV database. KV databases are typically lauded for the speed (Sadalage & Fowler, 2013)

(Sullivan, 2015) (Gudivada, Rao, & Raghavan, 2014) (Hecht & Jablonski, 2011) (Marz & Warren, 2015).

These characteristics influence the preprocessing ratings of the database types. While total effort may be broad defined, this information, including *LOC*, provides some insight into how much effort is required to complete the preprocessing for each of these database types.

Preprocessing Ratings

Table 14 contains the *Preprocessing* criterion performance ratings for the five database types studied. The ratings here are considered in the context of effort and workload required to perform all preprocessing tasks. Effort and workload were evaluated in this study primarily by the *Execution Duration* and *LOC*—with emphasis placed on LOC. The following analysis combines results from the literature review, table 10, and table 11 to develop the ratings presented.

Table 14. Preprocessing Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.50	3.00	0.33	7.00
Doc	2.00	1.00	4.00	0.25	7.00
Col	0.33	0.25	1.00	0.14	4.00
Graph	3.00	4.00	7.00	1.00	5.00
Rel	0.14	0.14	0.25	0.20	1.00

The data transformation required for the KV database type was slower than document, graph and relational types, but faster than the columnar type. This observation was consistent with the simplicity promised by this NoSQL model (Sadalage & Fowler, 2013) (Sullivan, 2015) (Hecht & Jablonski, 2011) (DeCandia, et al., 2007). Additionally, the KV type was slower than all other types for importing data. This could be attributed to Riak's method of storing data around its logical ring normally distributed in a multi-node cluster (Basho Technologies, Inc. , 2018)

(DeCandia, et al., 2007). While the overhead of this method may be tolerable for a distributed system, it was more noticeable in the single box environment. Moreover, KV required fewer LOC than column and relational types, the same as the document type, but more than the graph type for transforming data. For importing data, KV required more LOC than document and columnar, but less than graph and relational database types. Thus, reviewing the KV row of the *Preprocessing* performance rating table, KV-Doc and KV-Graph cells indicate that document and graph performed preprocessing better. In contrast, KV-Col and KV-Rel cells show that KV performed better.

The document type database's requirements for data transformation and importing were executed faster than the KV and column types, but slower than graph and relational types. This result meets expectations based on the nature of its aggregate model except for the KV example (Sadalage & Fowler, 2013) (Sullivan, 2015) (Hecht & Jablonski, 2011) (Chodorow, 2013). The difference for the KV model is suspected to be implementation-specific as previously described. In terms of LOC, the document type required the same number or fewer than KV, columnar, and relational, but the graph database used even less for the data transformation tasks. However, the document database required the least for importing data. Thus, in the document row of table 14, the Doc-Col and Doc-Rel cells indicate the document type performed better than these two types. In contrast, the graph database received a better *Preprocessing* performance rating as shown in the Doc-Graph cell.

The columnar database type's *Preprocessing* performance was also somewhat mixed. Its execution time for data transformation was the largest amongst the NoSQL types and nearly the largest for importing data. This database type also required the largest number of LOC for data transformation of the NoSQL types. However, the LOC for data import was relatively low. The

number of LOC required was less than the relational type for both task areas. The additional time and lines of code required were likely attributed to the way in which the data was modeled. Recall that tables were created to store the data required for each individual query as well as one to comprehensively store all of the flight data. Thus, multiple transformations were necessary to prepare and then import the log data for each table in this database. These steps could reasonably account for the additional time elapsed and lines of code during these phases. Thus, the remaining columnar ratings indicate its *Preprocessing* performance was worse than graph, but better than the relational type.

The graph database type's *Preprocessing* performance was the least straightforward to evaluate. For data transformation, the graph type was the clear NoSQL leader in terms of execution time and LOC with the lowest values for both categories. In fact, its number of LOC was the lowest amongst all types. Yet, the NoSQL types were categorically slower in execution for both data transformation and importing. Specifically, the graph type had the second fastest execution time for the *Import Data* task area for the NoSQL types. However, it required the largest number of LOC by a factor of ten for all types. In this case, the number of LOC does not provide the most accurate assessment of effort or workload because 10,038 LOC were automatically generated by some of the remaining 586 lines of code. Thus, for the remaining cell, i.e. Graph-Rel, the graph type was still assessed to perform better than the relational type.

LAT and SAT Observations

Additional observations can be made from Table 11. A simple review of the *Size on Disk* column illustrates the complete stored data set does not exceed 1 TB for any database; the largest database requires less than 40 GB to store all ten sets of data. Recall that Large Aggregate Transactions (LAT) criterion required a single aggregate to exceed 1 TB in size. Since the entire

data set is less than 1 TB, it is impossible for a single aggregate in this set to exceed this threshold. Therefore, no aggregates in this log data set would require the database to support LAT.

Table 15 displays the calculated average size of an aggregate in each database measured in kilobytes (kB). These values were determined by dividing the size on disk by the number of aggregates and then converting the result into kB. The average aggregate size for most of the NoSQL databases exceeds the threshold, i.e., 1 kB, for the Small Aggregate Transactions (SAT) criterion. However, the document and relational databases exhibited an average size that indicates SAT would apply. For the relational database, this is expected because its tables are normalized to minimize the amount of data they contain. Since normalization was part of the design for this study, the result is not unexpected.

Table 15. Calculated Average Size of Aggregates Per Database.

	Average Size of an Aggregate (kB)
KV (Riak)	7.97
Doc (Mongo)	0.47
Col (HBase)	1.78
Graph (Neo4j)	2.00
Rel (MySQL)	0.21

For MongoDB, this result was somewhat unexpected. In fact, MongoDB's Size on Disk result was also unexpectedly small because the other NoSQL databases were larger by an order of magnitude in all cases. However, MongoDB is designed for humongous data sets and uses a binary storage method to compress its data (Chodorow, 2013). Thus, this result is probably specific to the implementation and likely not reflective of document databases in general.

Queries

The *Queries* task area includes all activities associated with retrieving data from and updating data in the database. As previously described, a query involves one or more transactions involving one or more aggregates. In this study, queries were performed to retrieve data from the various database types according to a set of questions identified from the UAS user community. These questions were identified in Chapter III, figure 10.

Table 16 displays a summary of the results from the Queries task area of the simulation study. The *Execution Duration* column indicates the cumulative number of seconds lapsed while executing the queries resulting from the ten user community questions. Values in the *LOC* column display the total number of lines of code written and/or generated, during the study, to execute activities associated with this task area.

Table 16. Summary of Queries Task Area Results.

	T	Execution Duration (s)	LOC
KV (Riak)	Yes*	2.0@	70@
Doc (Mongo)	Yes	58.9	1,250
Col (HBase)	Yes	-	-
Graph (Neo4j)	Yes	1,718.5	38
Rel (MySQL)	Yes	3.8	84

* This capability is not enabled by default.
@Only 5 of the 10 queries were feasible and executed for this database.

Additionally, table 17 provides a summary of each database's native ability to perform a single query to retrieve the appropriate data necessary to answer each question identified by Chapter III, figure 10. The columns headers identify the question numbers and the row headers identify the databases. The cells in this table containing an 'x' indicate the database was capable of executing a query to retrieve and manipulate the data to answer the corresponding question. Empty cells

indicate that the question could not be answered by the native capabilities of the database and that third party software would be required to assist with the developing the answer.

Table 17. Database Query Success Summary.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
KV (Riak)	x	x					x	x	x	x
Doc (Mongo)	x	x	x	x	x	x	x	x	x	x
Col (HBase)										
Graph (Neo4j)	x	x	x	x	x	x	x	x	x	x
Rel (MySQL)	x	x	x	x	x	x	x	x	x	x

Many observations were made and applied to the evaluation of the database types for the relevant criteria. Recall, from Chapter III, table 4 that *LAT, SAT, RT, QC,* and *QO* criteria were all dependent on this task area. Since the *Queries* task area is the last remaining dependency for most of these criteria, their ratings will be discussed in this section. NOTE: The *LAT* ratings will be provided in the *Other Ratings* section because the study did not perform transaction involving large aggregates.

QC and QO Observations

First of all, not all databases could natively perform all queries to adequately answer the user community's questions. Specifically, HBase could not natively provide meaningful data to answer any of the questions. This database permitted queries to specify which column or columns were desired and which row in the table to retrieve. For example, if the data in the table designed for query #3 was requested, data would be retrieved for all samples (aggregates) for all flights. While some filtering is possible, significant limitations exist due to the lack of data typing enabled by HBase. The resulting data set was unmanageable, containing nearly 5 million rows, without additional third-party software to reduce or filter out certain rows. Without writing additional

functions to provide filtering, this quantity of data could not be reduced (Luo, Yu, Kevin, Gupta, & Spaggiari, 2018).

Likewise, even Riak KV's advanced queries which could search indexes and a range of values, did not support a query with multiple conditions, e.g., an airspeed with the landing gear down or a maximum altitude for each flight, such as those required for questions 3-6. Additionally, the advanced query capability is disabled in Riak KV by default. Furthermore, this feature requires that a schema is defined and that indexes are built on the elements of interest. Moreover, the Riak database, by default, expects results to be streamed to the requesting application. This works well for distributed systems, but less desirable for a single box environment because of the speed resulting from the locality of the data. While many of these characteristics are implementation-specific, they highlight the primary intent for simple, fast queries from the KV model.

Figure 19 displays a snippet of a query to retrieve data from Riak KV for question 2. This query is written using the Python language because Riak KV has no native shell or query language. Apache Solr provides the foundation for advanced searching for this database. Without Solr, searching would be limited to keys only (Apache Software Foundation, 2018). The first five lines are responsible for creating a connection to the Riak database. The line beginning with *results* calls a function to submit the query to the database and stores the results. The parameters passed to the *fulltext_search* function are the name of the database, and the query parameters. The query parameters, i.e., *'alt_agl_d:[15000 TO *]'*are Extractors based on Apache Solr. Additional code is required, but not included, to present the results in a meaningful way.

```
from get_db_connection import get_db_connection

# Get connection to Riak or exit
client = get_db_connection()
if client is None:
    exit()

results = client.fulltext_search('uas', 'alt_agl_d:[15000 TO *]')
```

Figure 19. Query #2 Written in Python 2.7 for Riak KV.

Ultimately, the KV and columnar results are generally expected because both types provide only simple querying capabilities (Sadalage & Fowler, 2013) (Sullivan, 2015) (Hecht & Jablonski, 2011) (DeCandia, et al., 2007). However, columnar database may be somewhat implementation specific because HBase is part of the Hadoop ecosystem which expects additionally layers, such as Hive, to be added to meet user needs. Though HBase provides a Java API to extend native capabilities, using this API would violates the intent of the study because this database natively provides *put, get,* and *scan* commands to store and retrieve data (White, 2015) (Spaggiari & O'Dell, 2016) (Apache Software Foundation, The, 2016) (Gates, 2016). Thus, even being designed with detailed knowledge of the expected queries, i.e. having some query omniscience, these databases were unable to execute some queries. Furthermore, these databases would also struggle to execute unexpected queries requiring the same complexity, e.g. specifying multiple conditions, searching for values within a specified range, or performing a mathematical operation on an aggregation[1] of values like determining the maximum value of a particular element or accumulating a total. Finally, these results make the values for KV database type in table 16 less meaningful because neither the elapsed time nor LOC reflects an outcome spanning completion of the expected tasks.

[1] Here aggregation is used in the traditional sense to mean multiple values are grouped together to produce a singular value.

In contrast to the KV and columnar databases which could not perform the queries well, the remaining database types were able to perform all queries to adequately answer the specified questions. Reviewing the LOC for the document, graph, and relational databases provides some insight into the queries and their complexity. The queries for the graph database type required the fewest LOC. This resulted for two reasons: relationships are native to the aggregates and the syntax for identifying relationships is simple. Specifically, the relationships are wholly contained in the edge-type aggregates rather than developed by traditional relational database joins.

Figure 20 and figure 21 present query #2 written in Neo4j's Cypher Query Language (CQL) and MySQL's Structured Query Language (SQL), respectively to illustrate both the similarities and difference in the languages. Query #2 corresponds to user question #2 which requests dates, times, and flights during which the aircraft exceeded 15,000 AGL. In this figure, the CQL line beginning with *MATCH* performs the equivalent *FROM/JOIN* operations in SQL. Specifically the CQL engine will identify the *Sample* nodes, s, which have the *COLLECTED_FOR* relationship/edge, r, with *Flight* nodes, f. For MySQL, query engine will join the *flight* and *navigation* tables on the *flight_id* key and join the *navigation* and *performance* tables on the composite *flight_id* and *sample_id* key to denormalize this data. The WHERE clause in both languages identifies the element, *alt_agl*, and the range, greater than 15000, for which data should be retrieved. The RETURN and SELECT clauses in each language identify which elements (attributes) should be extracted from the aggregate (denormalized table). The ORDER BY clause enables the extracted data to be sorted in ascending order by the elements (attributes) identified.

```
MATCH (f:Flight)<-[r:COLLECTED_FOR]-(s:Sample)
WHERE s.alt_agl >15000
RETURN f.flight_id_text, r.gps_time, r.gps_week, s.alt_agl
ORDER BY f.flight_id, r.gps_time;
```

Figure 20. Query #2 Written in Neo4j's Cypher Query Language (CQL).

```
SELECT flight.flight_id_text, navigation.gps_week,
navigation.gps_time, performance.alt_agl

FROM flight
JOIN navigation ON flight.flight_id = navigation.flight_id
JOIN performance ON navigation.sample_id = performance.sample_id
AND performance.flight_id = navigation.flight_id

WHERE performance.alt_agl > 15000

ORDER BY flight.flight_id, navigation.gps_time;
```

Figure 21. Query #2 Written in MySQL's Structured Query Language (SQL).

MongoDB provides the same capabilities using its shell to interpret JavaScript to execute queries. One limitation encountered was the query had to be re-written for each collection. It should be acknowledged that Mongo does permit cross-collection queries using a left-outer join via it's $lookup operator (MongoDB, Inc., 2018). However, for the UAS log data, there is nothing to join between flights. Since the data was split by flights into collections, this required ten nearly identical queries to be written to retrieve the results to answer question #2. Thus, the LOC for MongoDB was much larger for the *Queries* task area. A different schema design, which does not use collections to separate data between flights, would reduce the number of lines to retrieve the data. However, this data would then need to be separated by a third party application.

Figure 22 displays the query written in JavaScript for MongoDb to provide an answer for question #2, i.e. query #2. The first line identifies the collection in which to search. The second line indicates the element, *alt_agl*, and the range of interest, greater than 15,000, for which data should be retrieved. The following line is similar to the SQL *SELECT* statement signifying the

elements which should be retrieved. The last three lines inform the query engine the data should be sorted by *_id* and *gps_time*.

```
db.A398_01_G505_010810.find(
    { alt_agl : {$gte:15000}},
    { calculated_utc_time:1, alt_agl:1, gps_week:1, gps_time:1 }
).sort(
    {_id:1, gps_time:1} // sort results in ascending order by _id, then gps_time
);
```

Figure 22. Query #2 Written in JavaScript for MongoDB's shell.

The graph, relational, and document database types provide similar capabilities to search for, identify, retrieve, and sort the data of interest. These databases also provide for multiple mechanisms to search for and retrieve data of interest to provide flexible and powerful query performance. The KV and columnar database types support only limited queries and require significant foresight during their design to identify types of queries will be executed. In contrast, these types were less flexible in the ways in which they support data retrieval.

QC Ratings

Table 18 presents the QC criterion ratings results for the database types studied. The KV and columnar databases claim similar performance in terms of data retrieval, but in this study Riak KV performed better than the selected HBase implementation. However, since the ratings are intended to provide generalizations for each type, these two will be rated as equal to avoid bias resulting from implementation-specific issues since other databases of this type support these operations (ScyllaDB, 2018) (DataStax, Inc., 2018). In contrast, document, graph, and relational performed and support complex queries better than KV. Relational was exceptionally better, followed by graph and then document database types. Thus, the ratings in the KV-Doc, KV-Graph, and KV-Rel cells reflect these observations. The document type outperformed the columnar type for this

criterion but was not as effective as graph or relation as reported in the Doc-Col, Doc-Graph, and Doc-Rel cells. As discussed in the observations section, the graph type was substantially better than the columnar type and relational was even better. The ratings in the Col-Graph and Col-Rel cells identify this outcome. Finally, while the graph type did require fewer LOC to perform the queries, its extensive execution time cannot be overlooked. Therefore, the Graph-Rel cell rates the relational type as performing better.

Table 18. QC Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.20	1.00	0.13	0.11
Doc	5.00	1.00	5.00	0.22	0.20
Col	1.00	0.20	1.00	0.13	0.11
Graph	8.00	4.50	8.00	1.00	0.50
Rel	9.00	5.00	9.00	2.00	1.00

QO Ratings

Table 19 contains the results of the QO criterion assessment for the database types in this study. The ratings for this criterion assess how much knowledge of the queries must be known before the database is designed. Generally, the KV and columnar database types required the most forethought into query design before designing their schemas or aggregate structure. In contrast, the document, graph, and relational required less planning for queries during their design. The KV row in this matrix indicates that KV types require exceptional amounts of query planning compared to relational and graph database types. Additionally, the KV require significant planning compared to the document type and essentially the same planning effort as the columnar type. Additionally queries beyond those identified at schema/database design time would be significantly limited without redesigning the existing data storage structure (Sullivan, 2015) (Sadalage & Fowler, 2013) (Serra, Choosing technologies for a big data solution in the cloud,

2017). The document type result row shows that columnar databases require more query planning than document database types, but document types require more planning than graph and relational types. The columnar database results are the same as the KV results. Due to the limited nature of the queries for both of these types, much planning is required. Finally, the Graph-Rel cell indicates that a minor amount of additional planning is required for the graph database type.

Table 19. QO Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	7.00	1.00	8.00	9.00
Doc	0.14	1.00	0.14	2.00	3.00
Col	1.00	7.00	1.00	8.00	9.00
Graph	0.13	0.50	0.13	1.00	1.50
Rel	0.11	0.33	0.11	0.67	1.00

RT Observations

The Execution Duration column in table 16 and the literature review are the basis for the RT observations and ratings. Recall the Execution Duration was incomplete for KV database because only 5 of the 10 queries could be accomplished with this database. However, KV types are known for fast data retrieval times albeit for 'simple' queries (DeCandia, et al., 2007) (Sadalage & Fowler, 2013) (Sullivan, 2015). Despite HBase's performance in this study, columnar databases have also claimed fast query performance, particularly when run as in-memory applications (Hecht & Jablonski, 2011) (Abadi D. J., Query execution in column-oriented database systems (Doctoral dissertation), 2008) (Plattner, 2014) (Sadalage & Fowler, 2013). The MongoDB database performed better than Neo4j in terms of query execution time despite the large number of queries and lines of code required. Furthermore, Neo4j's query execution times were much longer than the relational. Thus, even though Neo4j lauds it query time performance over relational databases, this performance gain was not realized in the use case for this study. Future work may explore if a

different schema for the graph database could produce faster results. However, for this study, the results are as reported.

RT Ratings

Table 20 presents a matrix containing the Result Timeliness (RT) criterion ratings for the database types in this study. These ratings involved more reliance on the literature review than the previous ratings due to the lack of data from the study for the KV and columnar types.

Table 20. RT Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	4.00	5.00	5.00	4.00
Doc	0.25	1.00	5.00	2.00	1.50
Col	0.20	0.20	1.00	0.33	0.33
Graph	0.20	0.50	3.00	1.00	0.50
Rel	0.25	0.67	3.00	2.00	1.00

LAT & SAT Observations

Another observation which influences the Large and Small Aggregate Transaction criteria is obtained by reviewing the variables involved in the ten queries. Specifically, the resultant aggregates are considered; that is, each query transaction is assumed to produce an aggregate as the result of selecting one or more elements from the stored aggregates. Each of the resultant aggregates are involved in the transactions to extract the data requested through the query. For example, consider the query which determines the dates, times, and flights in which the aircraft exceeded 15,000 ft AGL—query #2. Examples queries for this data were shown previously. From these examples, it was determined that the following elements would be extracted from the appropriate aggregates: *flight_id_text, gps_time, gps_week,* and *alt_agl*.

Table 21 displays the elements, the length of their name, type, and the size of the type for their stored value. This table assumes that 2 bytes are required to store a character, doubles are stored using 8 bytes, and Boolean required 1 byte to be stored in each database type (Oracle Corporation, 2018) (White, 2015) (Sullivan, 2015). The *Name Length* refers to the name used to identify the element in the database. Thus, the length in bytes is twice the number of characters in the string. For example *alt-agl*,is 7 characters long, so 14 bytes are required to store it and *flight_id_text* is 14 characters long, so 28 bytes are required to store that name. The Data Type column identifies which data type is used to store the element. The Size column indicates how many bytes are needed to store this type. For strings, this is the number of expected characters multiplied by 2 bytes per character.

Using the example above for query #2 and the data from this table, the size of the resultant aggregate can be estimated. Each resultant aggregate for query #2 will need 76 bytes to store the element names and 62 bytes to store the values associated with the names for a total of 138 bytes excluding any overhead. Additionally, the largest resultant aggregate that could be produced using all of the elements of interest for these queries would require 344 bytes to store the names and 148 bytes to store the values for a total of 492 bytes excluding overhead. Thus, the largest resultant aggregate would be less than 1kB in size (excluding overhead) and therefore reach the threshold to be considered a SAT and not crossing the LAT threshold. It is possible these small resultant aggregates affected the performance of the databases studied. The performance ratings for SAT will include this consideration. Consequently, the LAT ratings will be largely based on the literature review.

Table 21. Variable Names and Sizes

Element Name	Name Length (Bytes)	Data Type	Size (Bytes)
Alt-AGL	14	double	8
Alt-GPS	14	double	8
Beta angle	20	double	8
Calculated UTC Time	38	string	40
Engine Speed	24	double	8
Flight_ID_Text	28	string	38
Fuel_Pump_Failed	32	Boolean	1
Gear up	14	Boolean	1
GPS time	18	double	8
GPS week	16	double	8
Link 1 Lost	22	Boolean	1
Link 2 Lost	22	Boolean	1
MAP Red Mismatch	32	Boolean	1
MAP Ylw Mismatch	32	Boolean	1
Pri AS	12	double	8
VSI	6	double	8
Total	344	N/A	148

LAT Ratings

Table 22 contains the LAT performance ratings for the five database types. The comparative study did not perform any transactions involving large aggregates, so these ratings are heavily influence by the literature review. Each row of the matrix will be discussed in the following paragraphs.

Table 22. LAT Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.50	0.50	3.00	3.00
Doc	2.00	1.00	1.00	4.00	4.00
Col	2.00	1.00	1.00	4.00	4.00
Graph	0.33	0.25	0.25	1.00	2.00
Rel	0.33	0.25	0.25	0.50	1.00

KV databases are appropriate for the real-time processing of "Big Data" and large aggregates, but they are not impervious to experiencing latency when aggregates are exceptionally large. In those cases, another type of database, such as document or columnar, is recommended. However, KV are still generally better than graph and relational databases for transactions involving large aggregates (Gudivada, Rao, & Raghavan, 2014) (DeCandia, et al., 2007) (Sullivan, 2015). The KV row in the matrix illustrates these points. Specifically, the KV-Doc and KV-Col cells rate document and columnar databases as performing LAT better which the KV-Graph and KV-Rel cells show KV is better.

Document and columnar database types are generally among the best choices for large aggregates. In distributed systems, document types can more easily shard data to other servers because of the inherent locality provided by its data model (Carpenter & Hewitt, 2016) (Anderson, Lehnardt, & Slater, 2010) (Chodorow, 2013). Similarly, columnar types can split their data by column family (Spaggiari & O'Dell, 2016) (Abadi D. J., Query execution in column-oriented database systems (Doctoral dissertation), 2008). However, this advantage is largely mitigated in a single box environment leaving them with a smaller locality advantage over relational and graph. The ratings in the Doc- and Col- rows reflects these findings.

Lastly, there is not clear advantage between graph and relational types in terms of performance for LAT. Even in distributed systems, graph database systems are not ideal for exceptionally large aggregates. This is because there is little theory or heuristics for efficiently separating or sharding the graph into equal partitions without knowledge of the data (Robinson, Webber, & Eifrem, 2015). Thus the Graph-Rel cell contains a 1.00 to represent equal performance.

SAT Ratings

Table 23 displays the results of the SAT criterion performance assessment, in a matrix, for the five database types in this study. The results are developed from the combination of the study's results and the literature review. The *Execution Duration* measurements and aggregate size estimates were among the results from the study influencing these ratings. The KV database type performs better than all four other types for transactions involving small aggregates (DeCandia, et al., 2007) (Hecht & Jablonski, 2011). Thus, the KV- cells indicate better performance for all comparisons. However, none of the ratings suggest much more than a moderate difference due to the limited data collected in the study. The document database is better than columnar types, but less effective than graph and relational types. Document type databases are particularly well-suited for large data sets and appear to sacrifice some performance for smaller data sets (Chodorow, 2013) (Han, E, Le, & Du, 2011). The Doc- cells in the matrix reflect these findings without exceeding a moderate difference in either direction. The columnar database types typically perform less well than graph and relational database with respect to transactions involving small aggregates (Hecht & Jablonski, 2011). The Col- cells show ratings that favor graph and relational types for this criterion. Lastly, the relational database type performed better for this criterion and this is captured in the Graph-Rel cell.

Table 23. SAT Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	3.00	3.00	3.00	4.00
Doc	0.33	1.00	3.00	0.33	0.50
Col	0.33	0.33	1.00	0.33	0.33
Graph	0.33	3.00	3.00	1.00	0.50
Rel	0.25	2.00	3.00	2.00	1.00

Transparency Observations

The last observation from the Queries task area is directly related to the Transparency criterion. All databases in the study provided some ability to view, search, or select individual aggregate elements through query transactions. The KV database type generally does not provide this capability. In fact, it is disabled by default in Riak KV and requires a defined schema which must identify and declare a data type for any elements which are expected to be "transparent" for queries. The other database types are expected to provide this capability by default and did in this study. However, HBase was the least accommodating to searching by element value. Its lack of data typing complicated the ability to search element values in an intuitive manner.

Transparency Ratings

Table 24 provides the ratings from the performance assessment of each database type for the Transparency criteria. The results of the study are mostly consistent with expectations from the literature review with two exceptions. The first exception involves the Riak KV database which unexpectedly provided transparency for retrieval queries. Thus, the KV- ratings were weighted slightly in favor of KV performance. That is 1/7 was used instead of 1/9 which would indicate absolute dominance by the compared types. The second exception was the difficultly with which HBase provided searching inside of aggregates. Therefore, the columnar results were weighted slight against this database type for comparisons with document, graph and relational types.

Table 24. Transparency Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.14	0.14	0.14	0.14
Doc	7.00	1.00	3.00	1.00	1.00
Col	7.00	0.33	1.00	0.33	0.33
Graph	7.00	1.00	3.00	1.00	1.00
Rel	7.00	1.00	3.00	1.00	1.00

Other Ratings

This section contains the ratings of the remaining criteria: *Manipulation* and *Plasticity*. The bulk of these ratings are derived from the available academic and practitioner literature. However, some insights from the study influenced these ratings, but did not produce metrics that were collected in the study.

Manipulation Ratings

Table 25 displays the results for the Manipulation criterion for the five database types of interest. This criterion highlights the fact that KV database types generally do not provide a means to update elements within stored aggregates. It is related to the transparency criterion which refers to the ability to read or search within elements in stored aggregates (Hecht & Jablonski, 2011). Thus, the KV type was assessed to perform poorly in the context of this criterion. All other types enable the capability to update elements within their aggregate structures. The ratings reflect these findings by indicating all types perform better than KV and perform equally when compared to all others.

Table 25. Manipulation Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.11	0.11	0.11	0.11
Doc	9.00	1.00	1.00	1.00	1.00
Col	9.00	1.00	1.00	1.00	1.00
Graph	9.00	1.00	1.00	1.00	1.00
Rel	9.00	1.00	1.00	1.00	1.00

Plasticity Ratings

Table 26 presents Plasticity criterion ratings matrix for the five studied database types. This criterion measures how easy an element can be added or removed from an existing aggregate. Generally, the KV and relational types are the least supportive of this capability. In fact, KV databases categorically do not allow either operation concerning element for a stored aggregate. The solution for the KV type is to remove the existing aggregate and replace it with another aggregate. For the relational type, this capability involves stopping the database to update the schema to remove or add an attribute to a table. The other types natively and easily support this feature (Sullivan, 2015) (Redmond & Wilson, 2012) (Sadalage & Fowler, 2013).

The ratings matrix reflects these findings. The KV-cells indicate poor performance for all comparisons including against the relational type. The other NoSQL types earned ratings scoring them higher than KV and relational for this capability. Lastly, the document, column, and graph types were rated equally amongst each other.

Table 26. Plasticity Performance Ratings Matrix.

	KV	Doc	Col	Graph	Rel
KV	1.00	0.11	0.11	0.11	0.50
Doc	9.00	1.00	1.00	1.00	5.00
Col	9.00	1.00	1.00	1.00	5.00
Graph	9.00	1.00	1.00	1.00	5.00
Rel	2.00	0.20	0.20	0.20	1.00

Performance Priorities

The performance ratings established in the previous sections will be used to develop what Saaty called local priorities in the AHP process (How to make a decision: The Analytic Hierarchy Process, 1990). To differentiate from other local priorities, these results will be called performance

priorities in this research effort. A straightforward way to calculate these priorities is to normalize the ratings by column and then calculate the average for the row. The result is a vector containing local, i.e., performance, priorities for the given matrix. Saaty & Tran demonstrated this process in their later work (On the invalidity of fuzzifying numerical judgments in the Analytic Hierarchy Process., 2007). The normalization approach is demonstrated in detail for the CAC criterion and applied to all 12 criteria.

Calculating the CAC Priority Vector Using Normalization

Table 27 presents the performance ratings for the five database types for the CAC criterion. Additionally, the total sum of each column has been calculated and included as the last row of this table. This sum is required to normalize the values for each column.

Table 28 contains the normalized values calculated using the sums presented in table 27. Specifically, the value in the KV-KV cell in Table 27, 1.00, was divided by the sum, 21.00. This result, 0.048, was recorded in the KV-KV cell in Table 28. This process was repeated for the Doc-KV, Col-KV, Graph-KV, and Rel-KV cells for the KV column. The process continued for the Doc, Col, Graph, and Rel columns until all ratings were normalized in this manner.

Table 27. CAC Ratings and Column Sums.

	KV	Doc	Col	Graph	Rel
KV	1.00	1.00	1.00	0.11	0.11
Doc	1.00	1.00	1.00	0.11	0.11
Col	1.00	1.00	1.00	0.11	0.11
Graph	9.00	9.00	9.00	1.00	1.00
Rel	9.00	9.00	9.00	1.00	1.00
Sum	21.00	21.00	21.00	2.33	2.33

Table 28. Normalized CAC Ratings and Priorities.

	KV	Doc	Col	Graph	Rel
KV	0.048	0.048	0.048	0.048	0.048
Doc	0.048	0.048	0.048	0.048	0.048
Col	0.048	0.048	0.048	0.048	0.048
Graph	0.429	0.429	0.429	0.429	0.429
Rel	0.429	0.429	0.429	0.429	0.429

Table 29 displays the results of the next step which produced the CAC performance priorities for each database type. These values were determined by calculating the average of the values in each row from Table 28. For example, the normalized values in the KV-KV, KV-Doc, KV-Col, KV-Graph, and KV-Rel cells, (0.048, 0.048, 0.048, 0.048, 0.048,) were averaged (0.048) and recorded in the first row of table 29. The remaining rows were averaged in the same way. The priorities for this criterion were easy to calculate and trace because the CAC performance ratings were relatively consistent; that is, they were either 1, 1/9, or 9.

In this priority table, larger values indicate better performance and a higher priority. Recall that CAC refers to the database capability to automatically update aggregates based on relationships with other aggregates. Since graph and relational database types were the types that supported this capability, it is not surprising they scored highest for this criterion and resulting priority.

Table 29. CAC Performance Priorities.

	Priorities
KV	0.048
Doc	0.048
Col	0.048
Graph	0.429
Rel	0.429

Combined Priorities for All 12 Criteria

Table 30 provides the results of this process for all 12 criteria. Specifically, each ratings matrix was normalized by column and the average was determined for each row to calculate the performance priorities. Like Table 29, larger values in this table indicated better performance by a database type for the particular criterion. Alone these results can loosely illustrate whether or not a database type performed well for a given criterion. However, the significance of these results is fully realized when they are combined with the priorities determined from the importance ratings to indicate which database(s) are suitable for the use case. The next few sections will discuss these topics.

Table 30. Performance Priorities for All 12 Criteria.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
KV	0.048	0.057	0.194	0.416	0.027	0.033	0.180	0.039	0.415	0.496	0.347	0.034
Doc	0.048	0.116	0.327	0.123	0.243	0.302	0.232	0.135	0.083	0.196	0.347	0.280
Col	0.048	0.031	0.327	0.073	0.243	0.302	0.080	0.039	0.415	0.055	0.089	0.126
Graph	0.429	0.233	0.076	0.186	0.243	0.302	0.466	0.326	0.050	0.105	0.184	0.280
Rel	0.429	0.562	0.076	0.203	0.243	0.062	0.042	0.461	0.038	0.149	0.033	0.280

Importance Ratings, Priorities, and Global Priorities

This section discusses the importance ratings obtained by UAS log data subject matters experts (SMEs). A total of nine inputs were received from the SMEs representing maintenance/engineering support/training, safety/accident investigation, and Flight Operations Quality Assurance (FOQA) perspectives. These inputs reflect the SMEs' importance ratings for each of the 12 criteria defined in this research. The inputs were recorded using the matrix shown in Chapter III, Table 5 and collected from users during site visits to their locations. A script containing the fundamental scale, criteria definitions, considerations, and examples was used for

the SMEs and is included in Appendix A. The research team was available to answer their questions about the definitions of the criteria but avoided influencing the SMEs judgements. Additionally, the SMEs were asked to explain the context for which they made their judgements and identify any assumptions they made which influenced their responses.

The following sections are organized by the SMEs' locations. The first part of each section contains a brief discussion of the mission tasks and SMEs expertise at the location. Next, the SMEs' importance ratings are provided. The ratings are followed by resultant priorities. The last part of each section will focus on the global priorities which indicate the most suitable database type determined by the application of AHP. The first location, *Charlie*, includes explanations of the calculations, but other locations results continue without repeating this information.

Location Charlie

Overview

Two SMEs from *Charlie* contributed importance ratings for this study. SME #1 was the enterprise-wide recognized "guru" for analyzing log data to support maintenance and operational issues. Other locations and personnel referred their challenging log data questions to this SME. This SME uses log data to troubleshoot operational and maintenance issues resulting from operator errors, hardware/software failures, and other system anomalies. SME #1 anticipated using historical log data to support analysis of problems encountered in the future. Additionally, this SME employs log data analysis to provide training to their unit personnel.

The second SME from this location was also an experienced log data analyst. SME #2 emphasized the importance of analyzing data on a disconnected, single box solution as their

experience proved operating environments are not always conducive to network-reliant solutions. This SME also expected to use log data to support root cause analysis of UAS problems.

Ratings

Table 31 and table 32 contain the importance ratings developed by SME #1 and SME #2 respectively. The row and column headings identify the criteria using the abbreviations described throughout this chapter. The diagonal of these matrices contains values colored in red to emphasize this symmetric boundary. As discussed, the SMEs provided inputs solely in the cells to the right of the diagonals. Thus, results to the left of the diagonal were determined by calculating the reciprocal of the corresponding cell on the right. For example, on the right side, cell DTE-SM contains 7.00 and was recorded directly from SME #1. Its symmetric left-side partner is the SM-DTE cell which contains 0.14 which is approximately 1/7—the reciprocal of the value in the DTE-SM cell.

Table 31. Importance Ratings Matrix - Location Charlie - SME #1.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	0.20	3.00	2.00	1.00	0.25	0.17	0.20	0.20	0.25	1.00	7.00
DTE	5.00	1.00	7.00	7.00	7.00	1.00	1.00	4.00	2.00	3.00	7.00	3.00
LAT	0.33	0.14	1.00	0.33	0.33	0.14	0.14	0.33	0.14	0.14	1.00	0.14
SAT	0.50	0.14	3.00	1.00	1.00	0.14	0.14	0.33	0.14	0.14	1.00	0.14
M	1.00	0.14	3.00	1.00	1.00	1.00	0.33	2.00	0.33	0.33	5.00	0.20
Pla	4.00	1.00	7.00	7.00	1.00	1.00	0.50	0.25	0.33	2.00	1.00	0.33
Pre	6.00	1.00	7.00	7.00	3.00	2.00	1.00	5.00	4.00	3.00	5.00	0.33
QC	5.00	0.25	3.00	3.00	0.50	4.00	0.20	1.00	0.33	4.00	5.00	2.00
QO	5.00	0.50	7.00	7.00	3.00	3.00	0.25	3.00	1.00	4.00	5.00	2.00
RT	4.00	0.33	7.00	7.00	3.00	0.50	0.33	0.25	0.25	1.00	5.00	0.20
SM	1.00	0.14	1.00	1.00	0.20	1.00	0.20	0.20	0.20	0.20	1.00	0.14
T	0.14	0.33	7.00	7.00	5.00	3.00	3.00	0.50	0.50	5.00	7.00	1.00

Table 32. Importance Ratings Matrix - Location Charlie - SME #2.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	0.11	9.00	5.00	1.00	3.00	0.14	0.11	7.00	0.33	0.13	0.20
DTE	9.00	1.00	9.00	7.00	0.50	5.00	0.33	0.33	9.00	0.33	0.50	0.50
LAT	0.11	0.11	1.00	0.11	0.11	0.11	0.11	0.11	0.50	0.11	0.11	0.11
SAT	0.20	0.14	9.00	1.00	0.33	0.33	0.20	0.14	3.00	0.20	0.33	0.33
M	1.00	2.00	9.00	3.00	1.00	0.50	0.33	0.50	5.00	0.33	0.20	0.50
Pla	0.33	0.20	9.00	3.00	2.00	1.00	0.50	0.25	7.00	0.33	0.20	0.20
Pre	7.00	3.00	9.00	5.00	3.00	2.00	1.00	0.50	9.00	7.00	5.00	2.00
QC	9.00	3.00	9.00	7.00	2.00	4.00	2.00	1.00	8.00	5.00	7.00	3.00
QO	0.14	0.11	2.00	0.33	0.20	0.14	0.11	0.13	1.00	0.20	0.20	0.14
RT	3.00	3.00	9.00	5.00	3.00	3.00	0.14	0.20	5.00	1.00	5.00	0.20
SM	8.00	2.00	9.00	3.00	5.00	5.00	0.20	0.14	5.00	0.20	1.00	0.33
T	5.00	2.00	9.00	3.00	2.00	5.00	0.50	0.33	7.00	5.00	3.00	1.00

Local Priorities

Using the same process for calculating performance priorities, importance priorities were determined for these inputs. Specifically, these ratings were normalized by column and then the average of the normalized values were calculated across each row to develop the importance priorities.

Figure 23 presents the resultant importance priority vectors from Location Charlie's SMEs. Part (a) contains the results from SME #1 and part (b) contains SME #2's results. In each part, the first column contains the list of criteria and the second column contains the calculated priorities which are normalized between 0 and 1. Additionally, the priorities are shaded using a white-to-green gradient from smallest to largest value to highlight the results.

These results are intermediate and provide limited insight when considered independently from the global priorities which have yet to be determined. However, reviewing the results can serve as a qualitative assessment of how well the judges understand the problem. For example, a preliminary review suggests SME #1 emphasized Data Typing Enforcement and Preprocessing while SME #2 appears to value Query Complexity. These observations are consistent with discussions between the research and the SME throughout the course of research. Specifically, SME #1 routinely emphasized that any solution should automatically ingest the data and retain a

high level of fidelity. Likewise, SME #2 expressed an appreciation for powerful and flexible data queries—which could be accomplished by a system that provides QC. Additionally, the low priority associated with LAT suggests the SMEs recognized their log data does not exceed this threshold and used the rating system appropriately. Observations like these provide confidence the AHP approach for this problem is appropriate.

Criteria	Importance Priority
CAC	0.061
DTE	0.166
LAT	0.015
SAT	0.021
M	0.047
Pla	0.077
Pre	0.166
QC	0.091
QO	0.127
RT	0.071
SM	0.022
T	0.136

(a) SME #1

Criteria	Importance Priority
CAC	0.050
DTE	0.092
LAT	0.009
SAT	0.029
M	0.061
Pla	0.053
Pre	0.169
QC	0.212
QO	0.013
RT	0.097
SM	0.094
T	0.121

(b) SME #2

Figure 23. Importance Priority Vectors - Location Charlie.

Another observation can be made by comparing the differences between SMEs' the priority vectors. The color gradients serve as a qualitative histogram where the bold green color corresponds to bins with a higher count and the white corresponds to bins with a low count; the criteria would correspond to categorical bins. Treating the results in this figure as such, it is easier to identify differences. In particular, there is a noticeable difference between the importance priorities for QO by both SMEs.

Global Priorities

Each judge has a global priority vector associated with their results. These global priorities were determined by combining the results of the performance and importance priorities.

Specifically, the importance priorities for each criterion are multiplied by the corresponding performance priority for each database type. These results are totaled for each database type to produce a vector of global priorities. For example, the performance priority, from table 30, for CAC for the KV database type is multiplied by SME #1's importance priority for CAC. Then this product is computed for the document, column, graph, and relational types. The calculations continue for all criterion until a new matrix of values is produced. Table 33 illustrates the intermediate results of these products. The last step is to sum the value across each row. These values are the SME's global priorities for each database type. Table 34 displays these results. The process is repeated for each SME.

Table 33. Matrix of Importance and Performance Products for SME #1.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
KV	0.0029	0.0095	0.0028	0.0088	0.0013	0.0025	0.0299	0.0036	0.0528	0.0351	0.0076	0.0047
Doc	0.0029	0.0193	0.0048	0.0026	0.0115	0.0232	0.0385	0.0124	0.0105	0.0139	0.0076	0.0381
Col	0.0029	0.0051	0.0048	0.0015	0.0115	0.0232	0.0133	0.0036	0.0528	0.0039	0.0019	0.0171
Graph	0.0260	0.0387	0.0011	0.0039	0.0115	0.0232	0.0772	0.0298	0.0064	0.0074	0.0040	0.0381
Rel	0.0260	0.0933	0.0011	0.0043	0.0115	0.0047	0.0069	0.0421	0.0049	0.0105	0.0007	0.0381

Table 34. SME #1's Global Priorities.

DB Type	Global Priority
KV	0.161
Doc	0.185
Col	0.142
Graph	0.267
Rel	0.244

Table 34 indicates a graph-type database, with the largest global priority, is the most suitable solution given the inputs provided by SME #1. According to this table, a columnar-type database is least suitable because has the lowest global priority associated with. The values in between these extremes vary in suitability. Table 35 provides SME #2's results for comparison. While the

resulting values are different for each SME, the conclusion is the same: the graph database type is most suitable.

Table 35. SME #2's Global Priorities.

DB Type	Global Priority
KV	0.154
Doc	0.205
Col	0.097
Graph	0.289
Rel	0.254

Location Hotel

Overview

The mission at this location is focused on providing training to UAS operators and maintainers. The three participating SMEs at this location, i.e., SME #3, SME #4, and SME #5, provide engineering support to operations, maintenance, and training activities. Part of their support involves analyzing UAS log data. SME #3 was a team lead and has less log data analysis experience than SME #4. SME #5 was relatively new to the team when the data was solicited from this team. In fact, SME #5 was the most junior log data analyst participating in this research effort. These SMEs would consider using a networked solution but understand the realities of working with the data on a single box.

Ratings

Table 36 presents the importance ratings obtained from SME #3. Unfortunately, SME #3 ran out of time during the visit and had not provided an update at the time of this writing. Thus, this data was incomplete. Additionally, these ratings contained values outside of the definitions in the fundamental scale—specifically it contained 0’s.

In this table, the cells highlighted in yellow denote the SME's inputs that were 0's. On the left side of the matrix's diagonal, these values would be 1/0—mathematically undefined, so these cells are left blank. The cells highlighted in red designate the cells for which the SME did not provide any rating. To avoid undefined values, these cells on the diagonal's left are also blank.

Table 36. Importance Ratings Matrix - Location Hotel - SME #3.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	1.00	0.20	0.20	0.50	1.00	0.20			0.33	1.00	1.00
DTE	1.00	1.00	0.20	0.20	0.50	1.00	0.20			0.33	1.00	1.00
LAT	5.00	5.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SAT	5.00	5.00		1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
M	2.00	2.00			1.00	1.00	0.20			0.20	0.20	1.00
Pla	1.00	1.00			1.00	1.00	0.20			0.20	0.20	0.20
Pre	5.00	5.00			5.00	5.00	1.00	7.00	7.00	7.00	7.00	7.00
QC							0.14	1.00	1.00	0.20	0.20	0.20
QO							0.14	1.00	1.00	0.20	0.20	0.20
RT	3.00	3.00			5.00	5.00	0.14	5.00	5.00	1.00	0.20	0.20
SM	1.00	1.00			5.00	5.00	0.14	5.00	5.00	5.00	1.00	0.20
T	1.00	1.00			1.00	5.00	0.14	5.00	5.00	5.00	5.00	1.00

These ratings are included as part of an effort to be straightforward and complete. Saaty's AHP methodology assumes the judge's ratings are complete, so there is no described approach for interpolation. In contrast to this set of ratings, the other two results were complete and conformed to the fundamental scale.

However, the data suggested that SME #4, despite instruction and discussion, took a more holistic approach to assigning ratings. Table 37 presents the ratings assigned by SME #4. Reviewing this data, it appears the judge performed column-wise comparisons as the cells in each column on the right side of the diagonal contain the same values. Without influencing the SME to change their answers, it was confirmed that this set of ratings reflects their evaluation of the criteria.

Table 37. Importance Ratings Matrix - Location Hotel – SME #4.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	0.50	9.00	9.00	9.00	0.25	9.00	0.14	9.00	0.33	0.17	0.20
DTE	2.00	1.00	9.00	9.00	9.00	0.25	9.00	0.14	9.00	0.33	0.17	0.20
LAT	0.11	0.11	1.00	9.00	9.00	0.25	9.00	0.14	9.00	0.33	0.17	0.20
SAT	0.11	0.11	0.11	1.00	9.00	0.25	9.00	0.14	9.00	0.33	0.17	0.20
M	0.11	0.11	0.11	0.11	1.00	0.25	9.00	0.14	9.00	0.33	0.17	0.20
Pla	4.00	4.00	4.00	4.00	4.00	1.00	9.00	0.14	9.00	0.33	0.17	0.20
Pre	0.11	0.11	0.11	0.11	0.11	0.11	1.00	0.14	9.00	0.33	0.17	0.20
QC	7.00	7.00	7.00	7.00	7.00	7.00	7.00	1.00	9.00	0.33	0.17	0.20
QO	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	1.00	0.33	0.17	0.20
RT	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	1.00	0.17	0.20
SM	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00	1.00	0.20
T	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	1.00

Table 38 displays the ratings matrix obtained from SME #5. No anomalous or unexpected ratings were detected upon preliminary review of this data set. It could be speculated that this respondent, as the junior analyst, had something to prove. Thus, their answers appear to be the most thoroughly considered of those collected from location *Hotel*.

Table 38. Importance Ratings Matrix - Location Hotel – SME #5.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	3.00	9.00	9.00	0.20	0.11	3.00	0.11	5.00	2.00	0.50	0.13
DTE	0.33	1.00	9.00	9.00	0.25	0.13	2.00	0.11	4.00	7.00	0.20	0.17
LAT	0.11	0.11	1.00	1.00	0.11	0.14	0.50	0.11	1.00	0.33	0.11	0.11
SAT	0.11	0.11	1.00	1.00	0.11	0.14	0.50	0.11	1.00	0.33	0.11	0.11
M	5.00	4.00	9.00	9.00	1.00	1.00	9.00	1.00	9.00	9.00	1.00	1.00
Pla	9.00	8.00	7.00	7.00	1.00	1.00	9.00	7.00	9.00	9.00	1.00	3.00
Pre	0.33	0.50	2.00	2.00	0.11	0.11	1.00	0.20	0.33	0.14	0.13	0.13
QC	9.00	9.00	9.00	9.00	1.00	0.14	5.00	1.00	9.00	8.00	3.00	1.00
QO	0.20	0.25	1.00	1.00	0.11	0.11	3.00	0.11	1.00	0.11	0.14	0.14
RT	0.50	0.14	3.00	3.00	0.11	0.11	7.00	0.13	9.00	1.00	0.50	0.25
SM	2.00	5.00	9.00	9.00	1.00	1.00	8.00	0.33	7.00	2.00	1.00	1.00
T	8.00	6.00	9.00	9.00	1.00	0.33	8.00	1.00	7.00	4.00	1.00	1.00

Local Priorities

Figure 24 displays the resultant priorities calculated using the normalization and averaging process previously described. Part (a) and (b) correspond to the ratings from SME's 4 and 5 respectively. As explained, the SME #3's results were not reliable because they were incomplete and did not use prescribed fundamental scale. Other observations may be made by reviewing SME

#4 and 5's resultant priority vectors. To the researcher, it seemed unusual that SM and T would be emphasized for the UAS log data, but it appears that these judges were in (at least) partial agreement on this matter. The green shading for these criteria suggests both SMEs valued these capabilities. Additionally, by treating the vectors as color-coded histograms again, similar patterns are apparent between these two SMEs results. That is, "areas" of green appear in similar locations for both result sets even though the bin counts are clearly not the same. Finally, it is apparent that SME #5, despite their junior status, understood that LAT plays an insignificant role for this use case.

Criteria	Importance Priority
CAC	0.076
DTE	0.081
LAT	0.058
SAT	0.043
M	0.031
Pla	0.075
Pre	0.021
QC	0.131
QO	0.012
RT	0.077
SM	0.179
T	0.216

(a) SME #4

Criteria	Importance Priority
CAC	0.058
DTE	0.057
LAT	0.013
SAT	0.013
M	0.146
Pla	0.225
Pre	0.017
QC	0.161
QO	0.017
RT	0.045
SM	0.118
T	0.131

(b) SME #5

Figure 24. Importance Priority Vectors – Location Hotel.

Global Priorities

The priorities shown in Figure 25 were calculated from the local priorities using the process described previously. The results from both SMEs at this location agree that a graph-type database would be most appropriate for their needs because the global priority for the graph type had the

largest value. In contrast, both results also suggest that a KV database would be least suitable because this database type had the smallest resultant global priority.

DB Type	Global Priority
KV	0.007
Doc	0.018
Col	0.012
Graph	0.023
Rel	0.022

(a) SME #4

DB Type	Global Priority
KV	0.109
Doc	0.232
Col	0.157
Graph	0.269
Rel	0.233

(b) SME #5

Figure 25. Global Priorities for Location Hotel SMEs.

Location Kilo

Overview

The mission at location Kilo involves investigations related to mishaps and overall safety. Only two SMEs participated from this location, i.e., SME #6 and SME #7. Both SMEs are involved with safety investigations and have experience analyzing the UAS log data using the available tools. SME #6 is the team leader and supervisor and assumes an analysis role commensurate with this title. Therefore, SME #7 has more "in the weeds" log data analysis experience, but is not considered a "junior" employee. In contrast to some other judges, SME #6 articulated that a networked solution would be ideal to support their operational needs concerning log data storage and retrieval. Both SMEs mentioned a desire to consolidate and use data from the existing two data repositories. Additionally, SME #7 observed that trend and historical analysis would be a desired outcome of using an automated system to manage the UAS log data.

Ratings

Table 39 and Table 40 present the importance ratings recorded by SME #6 and #7 respectively. The responses of these judges did not appear to contain any qualitative anomalies or suggest issues with their understanding of the rating system. Thus, these ratings were used to produce local priorities as part of the AHP methodology.

Table 39. Importance Ratings Matrix - Location Kilo - SME #6.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	0.50	4.00	4.00	2.00	1.00	0.17	0.50	6.00	4.00	2.00	2.00
DTE	2.00	1.00	2.00	2.00	0.50	0.25	0.13	1.00	6.00	2.00	2.00	4.00
LAT	0.25	0.50	1.00	1.00	0.25	0.25	0.13	0.25	0.25	0.25	0.50	0.50
SAT	0.25	0.50	1.00	1.00	0.25	0.25	0.13	0.25	0.25	0.25	0.50	0.50
M	0.50	2.00	4.00	4.00	1.00	0.50	0.50	0.50	0.50	0.50	0.25	0.50
Pla	1.00	4.00	4.00	4.00	2.00	1.00	0.50	4.00	4.00	4.00	4.00	4.00
Pre	6.00	8.00	8.00	8.00	2.00	2.00	1.00	6.00	8.00	4.00	4.00	6.00
QC	2.00	1.00	4.00	4.00	2.00	0.25	0.17	1.00	4.00	4.00	2.00	2.00
QO	0.17	0.17	4.00	4.00	2.00	0.25	0.13	0.25	1.00	0.25	0.25	0.50
RT	0.25	0.50	4.00	4.00	2.00	0.25	0.25	0.25	4.00	1.00	2.00	2.00
SM	0.50	0.50	2.00	2.00	4.00	0.25	0.25	0.50	4.00	0.50	1.00	0.50
T	0.50	0.25	2.00	2.00	2.00	0.25	0.17	0.50	2.00	0.50	2.00	1.00

Table 40. Importance Ratings Matrix - Location Kilo - SME #7.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	0.14	5.00	5.00	7.00	1.00	5.00	0.11	7.00	3.00	0.11	0.33
DTE	7.00	1.00	5.00	5.00	3.00	1.00	0.33	0.14	7.00	3.00	1.00	1.00
LAT	0.20	0.20	1.00	1.00	0.14	0.14	0.20	0.11	0.33	0.33	0.20	0.11
SAT	0.20	0.20	1.00	1.00	0.14	0.14	0.20	0.11	0.33	0.33	0.20	0.11
M	0.14	0.33	7.00	7.00	1.00	0.14	0.33	0.20	3.00	3.00	0.14	0.11
Pla	1.00	1.00	7.00	7.00	7.00	1.00	5.00	1.00	3.00	5.00	0.33	0.14
Pre	0.20	3.00	5.00	5.00	3.00	0.20	1.00	0.20	0.20	1.00	0.14	0.11
QC	9.00	7.00	9.00	9.00	5.00	1.00	5.00	1.00	7.00	7.00	1.00	1.00
QO	0.14	0.14	3.00	3.00	0.33	0.33	5.00	0.14	1.00	0.20	0.33	0.14
RT	0.33	0.33	3.00	3.00	0.33	0.20	1.00	0.14	5.00	1.00	0.33	0.14
SM	9.00	1.00	5.00	5.00	7.00	3.00	7.00	1.00	3.00	3.00	1.00	0.14
T	3.00	1.00	9.00	9.00	9.00	7.00	9.00	1.00	7.00	7.00	7.00	1.00

Local Priorities

Figure 26 provides the priorities resulting from the normalized column average process described previously. Part (a) and (b) contain the respective results for SME #6 and SME #7. A cursory review can observer difference between the pair of judges results. SME #6's results appear to emphasize fidelity (CAC and DTE) and import automation (Pre) which could be expected for a

supervisor and manager. In contrast, SME #7's results appear to value flexibility and power of queries (QC) and the storage structure (SM and T). This result is also expected for someone intimately familiar with the changes to the contents of the logs and variety of questions accumulating over time. Furthermore, these results indicate both SME's recognize that LAT is not particularly relevant for this system. This finding adds some confidence in the process thus far and its potential utility.

Criteria	Importance Priority
CAC	0.096
DTE	0.081
LAT	0.022
SAT	0.022
M	0.059
Pla	0.150
Pre	0.264
QC	0.093
QO	0.040
RT	0.067
SM	0.058
T	0.048

(a) SME #6.

Criteria	Importance Priority
CAC	0.077
DTE	0.099
LAT	0.013
SAT	0.013
M	0.045
Pla	0.097
Pre	0.049
QC	0.186
QO	0.032
RT	0.034
SM	0.127
T	0.229

(b) SME #7.

Figure 26. Importance Priority Vectors - Location Kilo.

Global Priorities

The priorities shown in Figure 27 were calculated using the previously described process. Again, the results from both SMEs at this location indicate a graph-type database would be most suitable for their requirements because the global priority for this type had the largest calculated value. In contrast, both results also suggest that a columnar-type database would be least appropriate because this type had the smallest global priority of the set in both cases.

DB Type	Global Priority
KV	0.152
Doc	0.208
Col	0.132
Graph	0.312
Rel	0.197

(a) SME #6

DB Type	Global Priority
KV	0.120
Doc	0.215
Col	0.118
Graph	0.275
Rel	0.271

(b) SME #7

Figure 27. Global Priorities for Location Kilo SMEs.

Location Mike

Overview

Two SMEs, SME #8 and SME #9, from *Mike* participated in this research effort. The mission at this location involved log data analysis of flight operations to identify trends and events that did or could result in operational problems. The results of this analysis work supported commanders across the enterprise in identifying shortcomings in training, techniques, tactics, and procedures. These SMEs use a substantial amount of automation to facilitate detection of known events and anomalies. The data used for their analysis is stored in one of the aforementioned repositories and accessible via a network. Additionally, these SMEs were both familiar with flight operations, databases operations, querying, and the analysis techniques required to extract information and analyze UAS log data. SME #8 had more experience than SME #9.

Ratings

Table 41 and table 42 display the importance ratings determined by SME #8 and #9 respectively. A preliminary review of their responses did not identify any unusual responses or indicate a lack of understanding concerning the rating system. Therefore, these results were used to calculate local priorities for the next step in AHP.

Table 41. Importance Ratings Matrix - Location Mike - SME #8.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	0.14	2.00	3.00	1.00	0.20	0.17	0.14	3.00	0.17	0.14	2.00
DTE	7.00	1.00	7.00	5.00	3.00	2.00	0.33	0.25	5.00	7.00	1.00	5.00
LAT	0.50	0.14	1.00	0.33	0.14	0.33	0.14	0.13	0.33	0.14	0.13	0.33
SAT	0.33	0.20	3.00	1.00	1.00	0.50	0.20	0.20	1.00	0.20	0.14	0.50
M	1.00	0.33	7.00	1.00	1.00	4.00	1.00	0.33	5.00	1.00	0.20	7.00
Pla	5.00	0.50	3.00	2.00	0.25	1.00	0.20	0.14	3.00	0.50	0.14	5.00
Pre	6.00	3.00	7.00	5.00	1.00	5.00	1.00	0.33	5.00	1.00	1.00	5.00
QC	7.00	4.00	8.00	5.00	3.00	7.00	3.00	1.00	7.00	0.20	0.20	3.00
QO	0.33	0.20	3.00	1.00	0.20	0.33	0.20	0.14	1.00	0.20	0.14	1.00
RT	6.00	0.14	7.00	5.00	1.00	2.00	1.00	5.00	5.00	1.00	1.00	7.00
SM	7.00	1.00	8.00	7.00	5.00	7.00	1.00	5.00	7.00	1.00	1.00	5.00
T	0.50	0.20	3.00	2.00	0.14	0.20	0.20	0.33	1.00	0.14	0.20	1.00

Table 42. Importance Ratings Matrix - Location MIke - SME #9.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
CAC	1.00	3.00	5.00	9.00	3.00	5.00	0.50	3.00	9.00	7.00	0.50	2.00
DTE	0.33	1.00	3.00	7.00	0.50	3.00	0.33	0.50	7.00	5.00	0.50	0.33
LAT	0.20	0.33	1.00	5.00	0.33	3.00	0.20	3.00	5.00	3.00	0.20	0.33
SAT	0.11	0.14	0.20	1.00	0.20	0.14	0.11	0.14	3.00	0.33	0.11	0.14
M	0.33	2.00	3.00	5.00	1.00	0.20	0.33	0.33	5.00	5.00	0.14	0.33
Pla	0.20	0.33	0.33	7.00	5.00	1.00	0.20	0.20	3.00	3.00	0.33	0.20
Pre	2.00	3.00	5.00	9.00	3.00	5.00	1.00	3.00	9.00	9.00	1.00	2.00
QC	0.33	2.00	0.33	7.00	3.00	5.00	0.33	1.00	7.00	7.00	0.33	0.33
QO	0.11	0.14	0.20	0.33	0.20	0.33	0.11	0.14	1.00	0.20	0.11	0.14
RT	0.14	0.20	0.33	3.00	0.20	0.33	0.11	0.14	5.00	1.00	0.14	0.14
SM	2.00	2.00	5.00	9.00	7.00	3.00	1.00	3.00	9.00	7.00	1.00	2.00
T	0.50	3.00	3.00	7.00	3.00	5.00	0.50	3.00	7.00	7.00	0.50	1.00

Local Priorities

Figure 28 presents the local priorities determined from the rating provided in the previous section. Part (a) and (b) contain the results calculated from the respective ratings from SME #8 and SME #9. Both SMEs appeared to value different aspects of data fidelity and consistency as indicated by their value of CAC and DTE. Additionally both sets of answers suggest sensitivity to the changing structure and contents of the log data through the resulting emphasis on M, Pla, and SM. Furthermore, neither SME from location *Mike* expected knowledge of all possible queries at the time of design (QO). Finally, SME #8 also determined that LAT was not appropriate for this use case. SME #9 resultant priorities did not clearly illustrate this understanding.

Criteria	Importance Priority
CAC	0.034
DTE	0.150
LAT	0.012
SAT	0.025
M	0.078
Pla	0.051
Pre	0.129
QC	0.162
QO	0.021
RT	0.133
SM	0.181
T	0.025

(a) SME #8.

Criteria	Importance Priority
CAC	0.147
DTE	0.072
LAT	0.057
SAT	0.015
M	0.059
Pla	0.052
Pre	0.179
QC	0.084
QO	0.012
RT	0.023
SM	0.178
T	0.121

(b) SME #9.

Figure 28. Importance Priority Vectors – Location Mike.

Global Priorities

The priorities shown in Figure 29 were determined using the previously described process. Once more, the results from both location *Mike* SMEs identify the graph database type as best suited for their needs due to the associated large global priority for this type. Additionally, the results agreed the columnar database type is least suitable for these SMEs.

DB Type	Global Priority
KV	0.194
Doc	0.210
Col	0.098
Graph	0.257
Rel	0.240

(a) SME #8

DB Type	Global Priority
KV	0.149
Doc	0.220
Col	0.114
Graph	0.298
Rel	0.219

(b) SME #9

Figure 29. Global Priorities for Location Mike SMEs.

Summary

This chapter presented the results of the methodology steps described in Chapter III. The simulation study and aggregate designs were presented first. The results of this study were used in conjunction with the literature review findings to establish performance ratings for each of the database types. These ratings were assessed using the AHP to develop performance priorities. The second half of this chapter described the importance ratings and resultant priorities calculated from the field study. Lastly, global priorities were calculated using the importance and performance results according to the third step of the AHP. The next chapter will summarize and discuss these global priorities.

V. Discussion

This chapter provides an overview of the effort to determine the suitability of relational and NoSQL systems as log data storage systems.

Interpretation of Global Priorities

Table 43 displays a summary of the global priorities calculated for all SMEs. This table provides the means to quickly compare the suitability results across locations and SMEs. The results suggest that one database type is most suitable for all locations. The results vary for subsequent alternatives. Additionally, the table does not include a result for location Hotel's SME #3 because the inputs from this SME could not be used to calculated local priorities. SME #3's results were discussed more thoroughly in Chapter IV.

Table 43. Summary of Global Priorities.

DB Type	C-SME #1	C-SME #2	H-SME #4	H-SME #5	K-SME #6	K-SME #7	M-SME #8	M-SME #9
KV	0.161	0.154	0.007	0.109	0.152	0.120	0.194	0.149
Doc	0.185	0.205	0.018	0.232	0.208	0.215	0.210	0.220
Col	0.142	0.097	0.012	0.157	0.132	0.118	0.098	0.114
Graph	0.267	0.289	0.023	0.269	0.312	0.275	0.257	0.298
Rel	0.244	0.254	0.022	0.233	0.197	0.271	0.240	0.219
Most Suitable Type	Graph	Graph	Graph	Graph	Graph	Graph	Graph	Graph

The top result, i.e. most suitable database type, for all SMEs was the graph type database. This type earned the highest global priority score across all locations and SMEs. This consistent result is an encouraging endorsement for the validity of the methodology because the users have relatively similar needs for log data storage and retrieval. Specifically, the mechanism should have produced similar outcomes for SMEs with identical needs. Thus, it was expected the results would agree for SMEs at the same location. However, this universal agreement was not necessarily anticipated.

Though the interpretation of the priorities agrees, the highest calculated global priority is different for each SME. Specifically, the highest scores in ascending order are 0.023, 0.257, 0.267, 0.269, 0.275, 0.289, 0.298, and 0.312. This difference is important and expected because SME's inputs were not in exact agreement because each input matrix resulted in different local priorities. Thus, the calculated global priorities would not be expected to match exactly.

Additionally, the second most suitable type was the relational database type as six of the eight SMEs second largest global priority corresponded to the relational type. The remaining two indicated the document type would be a suitable second choice. The document type was the third best result for the original six. Thus, the results for second and third most suitable varied between relational and document.

Furthermore, the columnar type database was solidly the least suitable choice among alternatives. Six SMEs global priorities agreed the columnar type was the worst solution for their needs. The remaining two results suggested key-value was the least appropriate. These results agreed that columnar types were next to last.

Another observation pertains to the smallest value of this set of top priorities. This value is more than two standard deviations smaller than the mean of these samples. This priority was calculated from the inputs of SME #4 whose importance ratings presented an unusual pattern. Recall this SME's ratings in each column of the input matrix were the same. Moreover, all this SME's global priorities were smaller than the results of the other SMEs. Yet, the interpretation of this judge's results matches those of the other judge from the same location. These two facts suggest the AHP approach and evaluation criteria are a robust mechanism for determining suitability of the data storage and retrieval systems.

Comments on the Methodology

Overall there was reasonable agreement between the resultant global priorities of the SMEs. Though, it was acknowledged the total number of participating judges was small. This number should be increased for a study which intends to identify a statistically relevant solution for a log data storage and retrieval system. Yet, since the purpose of this study was to determine as a proof of concept whether the evaluation criteria are suitable to compare popular database types, the number of participants is acceptable.

An aspect of Saaty's original AHP methodology was not included in this effort. The original AHP methodology included a means to assess the consistency of the inputs provided by each judge. However, in Saaty's later work involving AHP, he argued that inconsistent inputs do not invalidate the results. Thus, consistency was considered but not included in this project.

Future Work

In the future, a number of areas of this study could be extended. Additional use cases, besides the UAS log data, could be explored to augment the proof of concept for these criteria. For example, router logs on a small network or user access logs a group of buildings could be investigated. Both cases involve electronical log data generated by asynchronous events and could be used by personnel with different mission roles such as security, maintenance, and operations.

A sensitivity experiment could be accomplished for the criteria performance ratings. This would assess influential each criterion has with respect to the overall global ratings. This experiment could also be extended to include other database experts to assess the performance of each database type. As an experiment, this could augment the statistical rigor of the ratings and establish a well-accepted baseline of performance.

Another extension of this work could include using other database implementations. The databases selected for this research are only five of hundreds of options. A few other options, such as an in-memory version of a key-value database type, like Redis, or a simpler columnar database system like Cassandra would provide additional support for the performance results involving the for their respective types.

Finally, if six months or more time was available, a pilot study involving the graph database could be conducted at one of the units participating in the field study. Specifically, a graph database storage and retrieval application, like the one developed in the laboratory simulation, could be provided to the users at location Charlie. After a short period of training and usage, the users could provide feedback about whether the application met the storage and retrieval needs for their mission area. In this approach, a survey or user interviews could be employed to collect the data. This data could be used to further evaluate the validity of the proposed methodology and potentially tune the AHP inputs.

Conclusion

Ever since Codd formally defined the relational model in the 1970s, relational databases and its associated powerful SQL code have been the dominant standard by which data storage and retrieval have been measured. With the advent of modern, Big Data requirements, the relational model has been augmented with a variety of NoSQL type database options. However, for smaller, non-distributed applications, relational databases frequently remain the preferred database system. In many cases this is a prudent choice, but perhaps not always. In the UAS example studied, the inherent advantages of NoSQL databases yielded better results in a scaled-down single box environment.

Additionally, other data system practitioners have offered decision trees and various heuristics for selecting a data storage and retrieval system. However, these approaches typically lack the necessary rigor to choose a system beyond evaluating an issue with a discrete, e.g., binary or ternary, outcome such as data structure, the need for real-time event processing, or integrated analysis (Kosyakov, 2016) (Gessert, Wingerath, Friedrich, & Ritter, 2017) . The proposed approach, using AHP and the identified criteria, provides for pairwise comparison of multiple criteria to support a system decision.

Finally, the author believes this work could be easily extended to evaluate solutions for uses that do not include log data. This methodology was motivated by and applied to a use case involving log data. However, as the literature review illustrated, log data possesses characteristics that are similar to other data types and few characteristics that are unique. Additionally, log data follows the general data life cycle like most other data. Furthermore, none of the criteria, developed in this research, are specifically tied to any unique aspect of log data. Moreover, the evaluation methodology's primary constraint is small-scale applications. For these reasons, it is believed this evaluation methodology has applicability to a much wider range of single-box systems.

Appendix A. Importance Rating Script.

Single Computer Log Data Storage and Retrieval System Needs

The objective of this project is to identify data storage and retrieval systems that are best suited for the needs of <system name> log data users. Due to resource limitations and other constraints, only systems that run on a single computer will be considered. In the future, an Enterprise-wide solution may be evaluated. Twelve criteria were selected for evaluation and are shown in the input matrix on page two.

For this project, we ask you consider how you would like to use the data if it were available in a database stored on a single computer or laptop. Specifically, we request two tasks from you. The first task is to briefly describe the ideal use case in which you would store and retrieve log data from a database. The second task is to make comparisons regarding importance about each of the twelve criteria.

Task one may be accomplished on page two. Please provide an outline of the key aspects you would consider when accessing log data from a database. This outline should describe how you, personally, would use the data/database to accomplish tasks related to your job. You do NOT need to describe how the <system name> enterprise should use the data. Additionally, you can add to this description and clarify your outline as you work through task two. You may also include comments about why you rated one criterion more important than another.

For task two, an input matrix is provided to record your ratings on page three. The ratings for each comparison are recorded in the cells of the matrix. Specifically, your results are recorded by comparing the importance of the criterion in a matrix row to the criterion in identified in each

column. You may find it convenient to print the input matrix and fill it in using the other sections as reference. This process and the criteria are explained in detail in this document.

The remainder of this document provides pages to perform the tasks, outlines instructions for using the input matrix, explains the twelve criteria, and provides an example detailing how comparisons could be made.

Task 1. Briefly describe how you would like to store and retrieve log data to meet your mission needs.

Task 2. Input Matrix

	1. CAC	2. DTE	3. LAT	4. SAT	5. M	6. Pla	7. Pre	8. QC	9. QO	10. RT	11. SM	12. T
1. Cross-Aggregate Consistency (CAC)	1											
2. Data Typing Enforcement (DTE)		1										
3. Large Aggregate Transactions (LAT)			1									
4. Small Aggregate Transactions (SAT)				1								
5. Manipulation (M)					1							
6. Plasticity (Pla)						1						
7. Preprocessing (Pre)							1					
8. Query Complexity (QC)								1				
9. Query Omniscience (QO)									1			
10. Result Timeliness (RT)										1		
11. Structural Malleability (SM)											1	
12. Transparency (T)												1

Using the input matrix

The input matrix consists of rows and columns containing the criteria. To reduce space in this document, the full names of the criteria are provided in the rows and abbreviations are used as column headings. The relative importance of each criterion, with respect to the objective, is established using pairwise comparisons of criteria in each row to criteria in each column.

The comparisons are evaluated from left to right and then top to bottom in the matrix. Thus, Cross-Aggregate Consistency (CAC) is compared to itself, Data Typing Enforcement (DTE), Large Aggregate Transactions (LAT), and so on until CAC is compared to Transparency (T). Then DTE is compared to itself, LAT, and so on until DTE is compared to T. This process continues downward through the rows of the matrix. These criteria and key terms will be explained in the next section of the document.

The rating scale is provided in Table 44. Acceptable values for the comparisons are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, and 1/9. If the criterion in the row is more important to your ideal case than the criterion in the column, a whole number is recorded in the appropriate cell. If the column is more important than the row, a fractional number is recorded.

Table 44. Fundamental Scale

Importance intensity	Definition	Explanation
1	Equal importance	Two criteria, row a and column b, contribute equally to the objective.
3	Moderate importance of one over another	Experience and judgment moderately favor one criterion, row a, over another, column b.
5	Essential or strong importance	Experience and judgment strongly favor one criterion, row a, over another, column b.
7	Very strong importance	One criterion, row a, has demonstrated dominance in practice over another, column b.

9	Extreme importance	The evidence favoring one criterion, row a, over another, column b, is of the highest possible order.
2, 4, 6, 8	Intermediate values between the two adjacent ratings/judgements	Used when compromise is needed. For example, 6 can be used for the intermediate value between 5 and 7.
1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9	These values are used when you believe the criterion in the column, b, is more important than the criterion in the row, a.	

This process results in two notable outcomes. The first outcome is each criterion is compared to itself. These comparisons are already recorded for you as "1's" across the diagonal cells in the matrix. This comparison is mathematically important in the subsequent analysis and provided for you as an example comparison. Additionally, a complete evaluation of the matrix would result in redundant comparisons. For example, the comparison of CAC to DTE has a reciprocal comparison of DTE to CAC. To save time, these redundant comparisons have cells colored in black. You do not need to fill in these blackened cells. An example of a filled in table is shown below. Please note the numbers shown in this table are purely illustrative.

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
1. Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5	1/7	1/7	1/7	1	1/7
2. Data Typing Enforcement (DTE)		1	7	7	5	5	3	1	1 / 2	1	3	3
3. Large Aggregate Transactions (LAT)			1	1	1/5	1/5	1/5	1/5	1/7	1/7	1/5	1/3
4. Small Aggregate Transactions (SAT)				1	1/5	1/5	1/5	1/5	1/7	1/7	1/5	1/3
5. Manipulation (M)					1	1 / 2	1/3	1/3	1/5	1/7	2	1
6. Plasticity (Pla)						1	1 / 2	1	1/3	1/5	3	1
7. Preprocessing (Pre)							1	1	1 / 2	1/5	3	2
8. Query Complexity (QC)								1	1 / 3	1	3	2
9. Query Omniscience (QO)									1	1	5	3
10. Result Timeliness (RT)										1	5	5
11. Structural Malleability (SM)											1	1/3
12. Transparency (T)												1

Terminology

Aggregates and Elements

The concept of an *aggregate* is central to modern database (DB) systems. An aggregate is composed of one or more data elements and is considered an atomic unit to the DB during transactions. An element contains a value and is similar to a cell in a relational database table. For example, an aggregate could be a single sample of all variables (signals) collected at a particular moment in time. Likewise an element would be a single sample of a particular variable (signal) such as engine speed, altitude, or heading. Additionally, an atomic unit is the typical component manipulated during DB transactions.

Transaction

A transaction is a Create, Read, Update, or Delete (CRUD) operation performed on a single aggregate by the DB. Essentially, a transaction is an operation that stores, retrieves, or removes an aggregate in the database. For example storing a single sample of all signals, i.e. an aggregate, in the database would be a transaction.

Query

A query is a set of transactions performed by the DB on one or more aggregates and elements. For example, a query could be run to retrieve the coolant temperature samples for a particular mission. Alternatively, a query could be performed to identify aircraft tail numbers stored in the database. Furthermore, a query could delete all data for a particular mission.

Criteria Explanations

1. Cross-Aggregate Consistency (CAC)

DB ability to perform cascading updates to across two or more aggregates based on implicitly or explicitly defined relationships. A *cascading update* is a database term referring to changes performed automatically by the database when a change to one aggregate logically affects other aggregates due to a defined relationship.

This ability differs from manipulation. In a DB which supports CAC, the DB would identify the relationship between aggregates and perform the necessary updates. In contrast, in a DB which supports only manipulation, the DB executes update operations independently from any implicitly or explicitly defined relationships. That is, manipulation, by itself, ignores these relationships.

- Examples
    - Assume a mission was flown and the log data is loaded into a DB. The log data is stored using one aggregate for each 1Hz sample of data. The **aircraft tail number itself is stored in each aggregate**. Later, someone realizes the wrong tail number was recorded for the mission. This update affects over 200,000 samples and thus 200,000 aggregates in the database. Cross-aggregate consistency involves the process responsible for updating aggregates to the appropriate tail number.
    - Assume a mission was flown and the log data is loaded into a DB. The tail number of the aircraft is related to each mission it flew. The **relationship is stored** by the DB. Queries can determine which missions were flown by a particular tail number. Queries can also retrieve/update properties about relationships such as the start/end date/time of a mission flown by a particular tail.
- Considerations
    - When updates to stored data are required, should the DB be responsible for ensuring CAC?
        - If yes, CAC is ***more*** important than some criteria, because you need the DB to support enforcing this consistency.

        - If no, CAC is ***less*** important than some criteria, because your need can tolerate some inconsistencies or you can manage CAC in another way.
    - Do aggregates in your data model have significant relationships to other aggregates?
        - If yes, CAC is ***more*** important than some criteria, because your aggregates are related to each other and your DB should store, retrieve and maintain these characteristics.
        - If no, CAC is ***less*** important than some criteria, because your aggregates are NOT significantly related to each other.
    - Do you need to store, retrieve, and update properties about the relationship between aggregates?
        - If yes, CAC is ***more*** important than some criteria, because you need the DB to store, retrieve and maintain relationship properties.
        - If no, CAC is ***less*** important than some criteria, because you do NOT need the DB to store, retrieve and maintain relationship properties.

2. Data Typing Enforcement (DTE)

DB enforcement of data types during transactions. Data types may include floats, doubles, Booleans, strings and others.

- Example
    - Flap angle is recorded in degrees with floating-point precision by the UAS. Assume an acceptable reading is 5.045 degrees. In contrast, landing gear status is recorded as either up or down (Boolean: 1 or 0). DTE ensures that flap angle is stored in and retrieved from the DB as 5.045 rather than 5. Likewise, DTE also ensures that the landing gear status is stored and retrieved appropriately. These data types (float and Boolean) can be specified in schemas and enforced by some DBs.
- Considerations
    - Can the data type and precision be identified for each element/signal?
        - If yes, DTE is ***more*** important than some criteria, because your elements/signals have well-known types and precision.
        - If no, DTE is ***less*** important than some criteria, because your elements/signals may have unknown types or precision.
    - Does the DB need to retain and enforce data types for your elements/signals?

- If yes, DTE is ***more*** important than some criteria, because you stated it needs to.
- If no, DTE is ***less*** important than some criteria, because you stated it does NOT need to.

3. Large Aggregate Transactions (LAT)

DB can store, retrieve, and update large (>1TB) aggregates quickly (within a few seconds or less).

- Example
  - The combined size of all the signals (of interest) sampled in 1 second is greater than 1 terabyte (TB). "Of interest" refers to the signals you intend to store, retrieve, or update in the DB.
- Considerations
  - Is the combined size of all the signals of interest sampled in 1 second is greater than 1 TB?
    - If yes, then LAT is ***more*** important than some criteria, because your resultant aggregate is considered large (>1TB).
    - If no, then LAT is ***less*** important than some criteria, because your resultant aggregate is considered large (>1TB).

4. Small Aggregate Transactions (SAT)

DB can store, retrieve, and update small (<1kB) aggregates quickly (within a few seconds or less).

- Example
  - The combined size of all the signals (of interest) sampled in 1 second is less than 1 kilobyte (kB). "Of interest" refers to the signals you intend to store, retrieve, or update in the DB.
- Considerations
  - Is the combined size of all the signals of interest sampled in 1 second is greater than 1 kB?

- If no, then SAT is ***more*** important than some criteria, because your resultant aggregate is considered small (<1kB).
- If yes, then SAT is ***less*** important than some criteria, because your resultant aggregate is NOT considered small (<1kB).

5. Manipulation (M)

DB can <u>update elements within a single stored</u> aggregate.

This ability differs from CAC because manipulation executes update operations independently from any implicitly or explicitly defined relationships. That is, manipulation ignores these relationships. In a DB which supports CAC, the DB would identify the relationship between aggregates and perform the necessary updates.

- Example
  - Assume a mission was flown and the log data is loaded into a DB. At the start of the mission, the GPS signal was blocked and the first second of the mission has incorrect timing information. To update this information in the DB, you need to be able to change the appropriate timing elements within the stored aggregates. The manipulation property enables this operation.
- Considerations
  - Should the DB enable updates to elements, such as timing, in existing aggregates?
    - If yes, manipulation is ***more*** important than some criteria, because you need the DB to support updates to existing data.
    - If no, manipulation is ***less*** important than some criteria, because you do NOT need to update stored information.

6. Plasticity (Pla)

DB transactions can <u>add or remove elements within</u> one or more <u>stored aggregates</u>.

- Example

- Assume a mission was flown and the log data is loaded into a DB. Adequate GPS data was collected for the entire mission. Now you wish to use the existing GPS timing information to calculate the UTC time for each sample. Once the time is calculated, you want to store that with each sample. The plasticity property enables you to add this "Calculated UTC time" element to each aggregate.
- Assume a <system name> mission was flown and the log data is loaded into a DB. Later it is determined that the Engine Speed signal was included and contained corrupt data for this mission. You want to remove this element from all of the aggregates for this mission. The plasticity property enables you to remove elements from existing aggregates.

- Considerations
  - Should the DB support adding and removing elements in existing aggregates?
    - If yes, plasticity is ***more*** important than some criteria, because you the DB to allow you to add or remove elements to or from existing data.
    - If no, plasticity is ***less*** important than some criteria, because you do NOT need to add or remove elements to or from stored information.

7. Preprocessing (Pre)

The preprocessing and operations required to <u>store data</u> in the DB.

- Examples:
  - Deciding/selecting which variables to store in the DB.
  - Filtering the selected variables from the raw data.
  - Transforming filtered variables into a format suitable for loading.
  - Loading transformed data into the DB.
- Considerations:
  - Are you willing to perform preparation work before data can be loaded into the DB?
    - If no, then this criterion is ***more*** important than some criteria, because you expect the DB to facilitate data preparation, preprocessing, or load raw data.
    - If yes, then this criterion is ***less*** important than some criteria, because you do NOT expect the DB to facilitate this process.

8. Query Complexity (QC)

DB ability to <u>perform simple and complex queries</u>. A simple query retrieves data using a unique identifier or a range of identifiers and/or values. Complex queries involve additional conditions enforced on the operations.

- Examples
    - Simple:
        - Did the engine speed ever exceed 5500 RPM?
        - Did the engine speed exceed 5500 RPM on the mission flown today?
    - Complex:
        - Did the engine fan draw more than 33 Amps while the cooling fan was in auto?
        - What is the average coolant temperature during a mission for each aircraft tail number?
- Considerations
    - Do you need the database to perform sorting, grouping, mathematical calculations, and conditional searches on your data sets?
        - If yes, query complexity is ***more*** important than some criteria, because you need the DB to support sorting, grouping, calculations, etc.
        - If no, query complexity is ***less*** important than some criteria, because you do **NOT** need the DB to support sorting, grouping, calculations, etc.

9. Query Omniscience (QO)

User knowledge of the set of queries before database is used.

- Examples:
    - You possess knowledge of all the queries that will be run before ever using the database. You know every question that will ever be asked about the data. You never encounter unexpected events or circumstances which require unique investigations. (QO would be rated very important in this example.)
    - You know many of the questions that need to be answered by the data, but do encounter circumstances requiring unique investigations. (QO would be rated less important in this example.)

    - You cannot anticipate how the data will be used. (QO is not very important in this example.)

- Considerations:
    - Will you expect to add to or change queries?
        - If yes, then query omniscience is ***less*** important than some criteria, because you will need to update queries in the future.
        - If no, then query omniscience is ***more*** important than some criteria, because you will not need to update queries.
    - Do you expect to work with queries that you cannot change?
        - If yes, then query omniscience is ***more*** important than some criteria, because you will work with only a known or predefined set of queries.
        - If no, then query omniscience is ***less*** important than some criteria, because you will work with queries that will need to be changeable.
    - How well do you know what queries/data questions will be asked before system is designed? Do you know all the questions that will be asked about the data? Do you know all your queries now?
        - If your answer is "very well," "yes," and "yes", then this criterion is ***more*** important than some criteria, because you know all of your data questions yet and will NOT need to support additional queries/questions in the future.
        - Otherwise, this criterion is ***less*** important than some criteria, because you do NOT know all of your data questions yet and will need to support additional queries/questions in the future.

10. Result Timeliness (RT)

How quickly results are provided to the user after a query is started.

- Example
    - A query is described, run, and produces results within 10 seconds of starting to run.
- Considerations
    - Do you expect results within seconds?
        - If your answer is yes, then result timeliness is ***more*** important than some criteria because you expect timely results.
    - Are you willing to wait minutes for a result?

- If you answer is yes, then result timeliness is ***somewhat less*** important than some criteria, because you need the DB to be somewhat timely.

- Are you willing to wait hours for results?
  - If you answer is yes, then result timeliness is ***less*** important than some criteria, because timeliness is not important to your needs.

11. Structural Malleability (SM)

The ability to <u>add or remove "types" of aggregates</u> to or from the DB.

- Example
  - Aggregate types could include: engine subsystem, synthetic aperture radar system, pilot inputs, aircraft, etc. Aggregates types such as these would need to be added to the DB if an equivalent subsystem is added to the UAS. Specifically, if a new sensor system was added, a new aggregate type might need to be added to the DB to store and organize data for this system.
- Considerations
  - Are subsystems likely to be added to or removed from your platform?
    - If yes, then this criterion is ***more*** important than some criteria, because you need a solution to support these types of changes.
    - If no, then this criterion is ***less*** important than some criteria, because you do NOT need a solution to support these types of changes.

12. Transparency (T)

DB ability to store aggregates such that individual <u>elements within an aggregate can be viewed</u> and retrieved during read transactions.

- Example
  - Assume we design a DB using an aggregate model that contains all signals for a particular moment in time. Thus, engine speed, altitude, landing gear, tail number, timing information, and etc. are stored in each aggregate.
  - With a DB that supports transparency, we can search for and retrieve engine speed for a particular mission/sample.
  - Without transparency, searches are limited to retrieving the whole aggregate when only engine speed is desired.
- Considerations

- Do you care about retrieving individual elements, such as engine speed, from your data?
    - If yes, then this criterion is ***more*** important than some criteria, because you need to access individual data elements within aggregates.
    - If no, then this criterion is ***less*** important than some criteria, because you do NOT need to access individual data elements within aggregates.

An Example

This example is provided to illustrate how the input matrix could be filled in. Given a copy of the input matrix, we shall walk through a set of pairwise comparisons. As discussed, the process begins in the upper left portion of the matrix.

The first comparison between CAC and itself has already been recorded as "1." The "1" represents equal importance and is the expected results when comparing one criterion to itself. Again, all "self-comparisons have been recorded for you in the matrix.

Table 45. Empty Input Matrix for Example

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
1. Cross-Aggregate Consistency (CAC)	1											
2. Data Typing Enforcement (DTE)		1										
3. Large Aggregate Transactions (LAT)			1									
4. Small Aggregate Transactions (SAT)				1								
5. Manipulation (M)					1							
6. Plasticity (Pla)						1						
7. Preprocessing (Pre)							1					
8. Query Complexity (QC)								1				
9. Query Omniscience (QO)									1			
10. Result Timeliness (RT)										1		
11. Structural Malleability (SM)											1	
12. Transparency (T)												1

The next comparison is between CAC and DTE. First we should review the considerations of CAC. Let's assume that updates to data stored in the database are infrequent. For example, importing a set of mission data into the database a second time would be uncommon. Once the data for a particular mission is loaded, it does not need to change because the log data for a particular mission should not change after the mission is complete. The first CAC consideration leads us to "less important than some criteria" since we are managing CAC by assuming infrequent updates.

Let's also assume that we do not need to store relationships in the database. For example, we want to store the aircraft tail numbers with the missions (and mission data) they fly. Also, we never care to look across all the missions flow by a particular aircraft; we simply examine one mission at a time. Thus, there are no significant relationships that need to be stored explicitly because all of the data is "lumped together." The remaining criteria for CAC also suggests it is "less important than some criteria."

Now we examine DTE which is related to the database's enforcement of data precision. Let's assume that we want to ensure the precision recorded by the UAS is maintained appropriately in the database. For example, we want flap angle to be stored as 5.0459 rather than 5.05. Thus, the considerations for DTE lead us to believe it is "more important than some criteria."

At this point, we understand CAC and DTE well enough to compare them to each other. We believe that DTE is more important than CAC. We review the fundamental scale, Table 44 , to determine by how much, i.e., the importance intensity. Since we have no relationships to maintain

and few updates, CAC is particularly low. However, we have not "demonstrated dominance in practice," so the intensity is less than "7." Let's choose "5" because we are employing "experience and judgment." Since we are working through the CAC row and we decided it is less important than DTE, we need to record the value as "1/5" in the CAC vs. DTE cell. This indicates that CAC is less important than DTE by an intensity of 5. Alternatively, this is interpreted as DTE is strongly more important than CAC.

Table 46. Partial Matrix - CAC vs. DTE

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5										

The next comparison involves CAC versus LAT. Let's assume that a 10 hour UAS mission results in approximately 400 MB of log data. That means it collects about 40 MB per hour or ~0.01 MB per second. Thus an aggregate composed of all the data in a single 1 Hz sample should be approximately 0.01 MB or ~10 kB.

Reviewing the LAT considerations, we know the threshold is for 1 TB for an aggregate so, LAT is "less important." Probably very much less in many cases. In the last example we considered CAC and determined it was "less important than some criteria." Now we must make a determination about whether or not that holds true for LAT. If we choose to say that they are equally unimportant, we would record a "1." However, I believe that CAC is more important because the LAT threshold is significantly higher than the size of these UAS aggregates. It seems unlikely that the aggregates would ever approach 1 TB.

Now we must make a judgement with respect to intensity. Let's choose "3" to indicate that CAC is moderately more important than LAT. Thus a "3" is recorded in the CAC vs. LAT cell in the matrix.

Table 47. Partial Matrix - CAC vs. LAT

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3									

CAC and SAT are compared next. Using our previously defined assumptions, we believe aggregates will be approximately 10 kB. According to the SAT considerations, the SAT threshold is 1 kB, so it is "less important than some criteria." Since the aggregate is somewhat close to the SAT threshold it might be inaccurate to say CAC is moderately more important than SAT. An intermediate value of "2" should suffice. Thus we record a "2" in the CAC vs. SAT cell.

Table 48. Partial Matrix - CAC vs. SAT

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2								

CAC is compared with M next. We already assumed that updates to aggregates are infrequent, so we will extend that assumption to state that updates to elements within aggregates are also uncommon. For example, it is unlikely that we would need to adjust the value of the recorded coolant temperature for a single sample in a particular mission. Thus, the M considerations indicate it is "less important than some criteria."

Since we extended the original assumption for CAC to M, it seems appropriate to say that these criteria are equally important. Thus a "1" is recorded in the CAC vs. M cell.

Table 49. Partial Matrix - CAC vs. M

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1							

The next comparison is between CAC and Pla. Let's assume that we wish to add and element to certain aggregates after they have been stored in the database. For example, let's use "Calculated UTC Time" as described from the Plasticity example in the Details section. Simply put, we want to add this calculated timing value to all the aggregates for a particular mission.

The considerations for this criterion indicate that adding elements causes Plasticity to be more important than some criteria. Because we can expect that such an update would affect hundreds of thousands of samples, i.e., aggregates, we should consider its importance to be high. Since this has not been demonstrated in practice, we should choose an intensity less than "7." Let's choose the intermediate value of "6." We intend to indicate that Pla is more important than CAC, so this value is recorded as "1/6" in the CAC vs. Pla cell.

Table 50. Partial Matrix - CAC vs. Pla

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6						

Working further towards the right side of the matrix, CAC is compared to Pre next. Let's assume we want to spend the minimal amount of time and effort required to manipulate data before

it can be loaded into the DB. This effort should include development for any scripting and not just manual processes. With this assumption, the Pre considerations suggest it is more important than some criteria.

Let's say we really do not want to spend time on these processes, we can choose "5" to represent the importance intensity of Pre over CAC. We record "1/5" since we are actually comparing CAC to Pre.

Table 51. Partial Matrix - CAC vs. Pre

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5					

The next two criteria to be compared with CAC are QC and QO. Let's assume that we expect to run complex queries on the data and at this time we cannot comprehensively anticipate every query we will need to perform. For example, we might have a history of known issues we expect to examine. Specifically, we might need to determine if the engine fan drew more than 30 Amps while the cooling fan was in Auto. This is an example of a complex query. Additionally, future system upgrades prevent us from identifying every question that will be asked about the system or data.

The considerations for QC suggest it will be more important because we need to perform complex queries. Since we have some practical history with examining data and let's assume we believe complex queries are very important, we can choose "7." It will be recorded at "1/7" in the CAC vs. QC cell.

Similarly, we believe additional queries will need to be defined in the future so the QO considerations also indicate it is more important. Since we have experience with system upgrades changing what data is analyzed and we also believe this characteristic is very important, "7" is chosen again to represent the importance intensity. The CAC vs. QO cell will record this as "1/7."

Table 52. Partial Matrix - CAC vs. QC and CAC vs. QO

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5	1/7	1/7			

CAC and RT are compared next. Let's assume that we want to obtain query results within seconds of requesting data. The RT considerations guide us to believe it is more important than some criteria. Compared to CAC, it might be strong or very strong. If we decide that we have enough practical experience to understand that quick results matter, we should choose "7." This value would be recorded as "1/7" in the CAC vs. RT cell to indicate the importance of RT (fast results) is very strong compared to CAC.

Table 53. Partial Matrix - CAC vs. RT

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5	1/7	1/7	1/7		

CAC is compared to SM next. Let's assume that a new subsystem could be added to our UAS. For example, an additional sensing pod or weapon system could be integrated. These types of additions can affect how the aggregates are stored in the DB. The considerations for SM indicate

that if subsystems are likely to be added, then it is more important. Since it is possible to add these systems but occur infrequently, our assumption is similar to those we used for CAC.

Let's use the logic we applied to M for this criterion as well. Thus, we will record that CAC and M are equal by recording a "1" in the CAC vs. M cell.

Table 54. Partial Matrix - CAC vs. SM

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5	1/7	1/7	1/7	1	

The last comparison in the CAC row is against T. A reasonable assumption is that we will want to retrieve an individual element, i.e. signal, from the database. For example, we might need to ask for the coolant temperature for a mission. The considerations for T show that given our assumption, T is more important that some criteria.

Since we know with some practice and certainty that individual signals need to be viewed, we can choose a "7" for this criterion. Thus, a "1/7" is recorded in the CAC vs. T cell to indicate the important of T over CAC.

Table 55. Partial Matrix - CAC vs. T

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5	1/7	1/7	1/7	1	1/7

Table 56 presents the results of the pairwise comparisons for the CAC row. This approach of evaluating each criterion should be repeated for the remaining unshaded cells in the matrix. Your

results for the first row may use different assumptions that will probably result in different values. This is expected and encouraged.

Table 56. Input Matrix with CAC vs. Comparisons Completed

	CAC	DTE	LAT	SAT	M	Pla	Pre	QC	QO	RT	SM	T
1. Cross-Aggregate Consistency (CAC)	1	1/5	3	2	1	1/6	1/5	1/7	1/7	1/7	1	1/7
2. Data Typing Enforcement (DTE)		1										
3. Large Aggregate Transactions (LAT)			1									
4. Small Aggregate Transactions (SAT)				1								
5. Manipulation (M)					1							
6. Plasticity (Pla)						1						
7. Preprocessing (Pre)							1					
8. Query Complexity (QC)								1				
9. Query Omniscience (QO)									1			
10. Result Timeliness (RT)										1		
11. Structural Malleability (SM)											1	
12. Transparency (T)												1

Bibliography

Abadi, D. J. (2008). *Query execution in column-oriented database systems (Doctoral dissertation).* Boston: Massachusetts Institute of Technology.

Abadi, D. J. (2012, 02). Consistency tradeoffs in modern distributed database system design. *Computer-IEEE Computer Magazine*, pp. 37-42.

Abadi, D. J., Boncz, P. A., & Harizopoulos, S. (2009). Column-oriented database systems. *Proceedings of the VLDB Endowment* (pp. 1664-1665). ACM.

Abadi, D. J., Boncz, P. A., Harizopoulos, S., Idreos, S., & Madden, S. (2013). The design and implementation of modern column-oriented database systems. *Foundations and Trends in Databases, 5*(3), pp. 197-280.

Allen, S. T., Jankowski, M., & Pathirana, P. (2015). *Storm Applied.* Shelter Island: Manning Publications Co.

Amazon Web Services. (2016). *What is Big Data.* Retrieved 12 01, 2016, from Amazon Web Services: https://aws.amazon.com/big-data/what-is-big-data/

Anderson, J. C., Lehnardt, J., & Slater, N. (2010). *CouchDB: The Definitive Guide.* Sebastopol: O'Reilly Media, Inc.

Anthony, S. (2013, 07 19). *Microsoft now has one million servers – less than Google, but more than Amazon, says Ballmer*. Retrieved from Extreme Tech: http://www.extremetech.com/extreme/161772-microsoft-now-has-one-million-servers-less-than-google-but-more-than-amazon-says-ballmer

Apache Software Foundation. (2018, 04 03). *Apache Solr*. Retrieved from Apache: http://lucene.apache.org/solr/

Apache Software Foundation, The. (2016, 10 11). *Welcome to Apache(tm) Hadoop(R)!* Retrieved 11 29, 2016, from Apache Hadoop: http://hadoop.apache.org/

Athow, D. (2016, 11 29). *Top 10 best servers of 2016 for SMBs.* Retrieved from TechRadar.Pro: http://www.techradar.com/news/computing/servers/the-top-10-servers-for-business-1100986

Baker, M. (2011, 07 13). *Search vs. Query*. Retrieved from Every Page is Page One: http://everypageispageone.com/2011/07/13/search-vs-query/

Ball, A. (2012). *Review of data management lifecycle models.* Bath: University of Bath.

Basho Technologies, Inc. (2018, 03 25). *Riak KV Key/Value Modeling*. Retrieved from Basho Documents: https://docs.basho.com/riak/kv/2.0.5/developing/key-value-modeling/

Basho Technologies, Inc. . (2018, 07 13). *Clusters*. Retrieved from Riak KV:Basho docs: http://docs.basho.com/riak/kv/2.2.3/learn/concepts/clusters/

Boston University. (2017). *Data Life Cycle*. Retrieved 08 02, 2017, from Boston University: https://www.bu.edu/datamanagement/background/data-life-cycle/

Bracket, M., & Kempe, S. (2013, 07 16). *Data Schemas and Data Structures 1*. Retrieved 10 2017, from DataVersity: http://www.dataversity.net/data-schemas-and-data-structures-1/

Bradbury, D. (2017, 03 11). *Database Schema vs. Data Structure*. Retrieved 10 2017, from StackOverflow: https://stackoverflow.com/questions/42738719/database-schema-vs-data-structure

Brewer, E. (2012, 02). CAP twelve years later: how the "rules" have changed. *Computer*, pp. 23-29.

Carpenter, J., & Hewitt, E. (2016). *Cassandra: The Definitite Guide.* Sebastopol: O'Reilly Media, Inc.

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record, 39*(4), 12-27.

Chamberlin, D. D., & Boyce, R. F. (1976). SEQUEL: A structured English query language. *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access, and control* (pp. 249-264). New York: ACM.

Chamberlin, D. D., Astrahan, M. M., Eswaran, K. P., Griffiths, P. P., Lorie, R. A., Mehl, J. W., . . . Wade, B. W. (1976). SEQUEL 2: A unified approach to data definition, manipulation, and control. *IBM Journal of Research and Development*, 560-575.

Chen, H., Kazman, R., & Haziyev, S. (2016). Agile Big Data Analytics Development: An Architecture-Centric Approach. *System Sciences (HICSS), 2016 49th Hawaii International Conference on* (pp. 5378-5387). IEEE.

Chen, M., Shiwen, M., & Liu, Y. (2014). Big data: a survey. *Mobile Networks and Applications*, pp. 171-209.

Chodorow, K. (2013). *MongoDB: The Definitive Guide.* Sebastopol: O'Reilly Media, Inc.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 377-387.

Connolly, T., & Begg, C. (2005). *Database Systems: A Practical Approach to Design, Implementation, and Management.* New York: Addison-Wesley.

Copeland, G. P., & Khoshafian, S. N. (1985). A Decomposition Storage Model. *ACM SIGMOD Record*, 268-279.

Dash, J. (2013, 09 18). *RDBMS vs. NoSQL: How do you pick?: An earnest argument for NoSQL...from an RDBMS veteran of IBM and Oracle.* Retrieved from ZDNet: http://www.zdnet.com/article/rdbms-vs-nosql-how-do-you-pick/

DataStax, Inc. (2018, 07 15). *SELECT | CQL for Cassandra*. Retrieved from DataStax: https://docs.datastax.com/en/cql/3.1/cql/cql_reference/select_r.html

DeCandia, G., Hastrun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., . . . Vogels, W. (2007). Dynamo: Amazon's Highly Available Key-value Store. *ACM SIGOPS Operating Systems Review*, 205-220.

Denning, P. J. (2005). The locality principle. *Communications of the ACM*, 19-24.

Dixon, J. (2010, 10 14). *Pentaho, Hadoop, and Data Lakes*. Retrieved from James Dixon's Blog: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/

Edlich, S. (2016, 11 21). *NoSQL Databases*. Retrieved from NoSQL Databases: http://nosql-database.org/

Eick, S. G., Nelson, M. C., & Schmidt, J. D. (1994). Graphical analysis of computer log files. *Communications of the ACM, 37*(12), pp. 50-56.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems.* Boston: Pearson.

Evans, E. (2011). *Domain-Driven Design: Tackling Complexity in the Heart of Software Engineering.* Westford: Pearson Education, Inc.

Fan, Y. T., & Wang, S. J. (2010). Intrusion investigations with data-hiding for computer log-file forensics. *Future Information Technology (FutureTech), 2010 5th International Conference on* (pp. 1-6). IEEE.

Fayyad, U., & Stolorz, P. (1997, 04 06). Data mining and KDD: promise and challenges. *Future Generation Computer Systems*, pp. 99-115.

Federal Deposit Insurance Corporation. (2014, 04 20). *FDIC Law, Regulations, Related Acts: 2000 Rules and Regulations*. Retrieved 07 27, 2017, from FDIC: https://www.fdic.gov/regulations/laws/rules/2000-6400.html#fdic2000part344.4

Fowler, M. (2015, 02 05). *Data Lake*. Retrieved 07 25, 2017, from Martin Fowler: https://martinfowler.com/bliki/DataLake.html

Fox, A., & Brewer, E. A. (1999). Harvest, yield, and scalable tolerant systems. *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems* (pp. 174-178). IEEE.

Gates, A. (2016, 11 13). *Hive Transactions.* Retrieved from Apache: https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions

Ge, R., Feng, X., Song, S., Chang, H. C., Li, D., & Cameron, K. W. (2010). Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 658-671.

Gessert, F., Wingerath, W., Friedrich, S., & Ritter, N. (2017). NoSQL database system: a survey and decision guidance. *Computer Science-Research and Development*, 353-365.

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News, 33*(2), pp. 51-59.

Gray, J. (1981). The transaction concept: virtues and limitations. *Proceedings of the 7th International Conference on Very Large Database Systems* (pp. 144-154). Cannes: ACM.

Griffiths, G., Brito, M., Robbins, I., & Moline, M. (2009). *Reliability of two REMUS-100 AUVs based on fault log analysis and elicited expert judgment.* Autonomous Undersea Systems Institute .

Gudivada, V. N., Rao, D., & Raghavan, V. V. (2014). NoSQL systems for big data management. *2014 IEEE World Congress on Service (SERVICES)* (pp. 190-197). IEEE.

Haerder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 287-317.

Han, J., E, H., Le, G., & Du, J. (2011). Survey on NoSQL Database. *2011 International Conference on Cloud and Service Computing (CSC)* (pp. 336-341). IEEE.

Headquarters, United States Air Force. (2014). *RPA Vector: Vision and Enabling Concepts 2013-2038.* Washington DC: Headquarters, United States Air Force.

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

Hoffer, J. A., Venkataraman, R., & Topi, H. (2016). *Modern Database Management.* Pearson.

IBM. (2017, 08 09). *Liberty: Storing transactions logs in a relational database*. Retrieved 08 28, 2017, from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_store_logs_in_rdb.html

IBM. (2017, 07 18). *Three-tier architectures*. Retrieved from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/covr_3-tier.html

IBM. (2017). *Three-tiered client/server architecture.* Retrieved from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SSAL2T_8.1.0/com.ibm.cics.tx.doc/concepts/c_three_tierd_clnt_sevr_arch.html

IBM Knowledge Center. (2017, 07 14). *The log files.* Retrieved from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q004020_.htm

IDC. (2014, 04). *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things.* Retrieved from EMC: http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

IEEE. (2015). *ISO/IEC/IEEE 15288: Systems and software engineering - System life cycle processes.* IEEE.

Inmon, W. (2002). *Building the Data Warehouse* (3rd ed.). New York: John Wiley & Sons, Inc.

ISO/IEC/IEEE. (2011). *ISO/IEC/IEEE 42010:2011(E) Systems and software engineering - architecture description.* ISO/IEC/IEEE.

Jain, A. (2016, 09 17). *The 5 Vs of Big Data*. Retrieved from IBM: https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/

John, T., & Misra, P. (2017). *Data Lake for Enterprises.* Birmingham: Packt.

Johnson, D. B., & Zwaenepoel, W. (1987). Sender-based message logging. *Rice University, Department of Computer Science*, 14-19.

Joshi, A. (2016). *Julia for Data Science.* Birmingham: Packt>.

Josuttis, N. M. (2012). *The C++ standard library: a tutorial and reference.* Addison-Wesley.

Kleppmann, M. (2017). *Designing Data-Intensive Applications.* Sebastopol: O'Reilly.

Kosyakov, I. (2016, 08 30). *Big Data Solutions Decision Tree*. Retrieved from Business Excellence: https://biz-excellence.com/2016/08/30/big-data-dt/

Kreps, J. (2013, December 16). *The Log: What every software engineer should know about real-time data's unifying abstraction.* Retrieved from Linked In (R) Engineering: https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

LabSat. (2016, 04 18). *GPS Time Converter*. Retrieved from LabSat: https://www.labsat.co.uk

Laney, D. (2001). *3D Data Management: Controlling Data Volume, Velocity, and Variety.* META Group Research Note 6.

Larose, D. T., & Larose, C. D. (2015). *Data Mining and Predictive Analytics.* Hoboken: John Wiley & Sons, Inc.

Leavitt, N. (2010, 01 25). Will NoSQL Databases Live Up to Their Promise? *Computer*, pp. 12-14.

Lee, I., Iyer, R. K., & Tang, D. (1991). Error/failure analysis using event logs from fault tolerant systems. *Fault-Tolerant Computing, 1991. FTCS-21. Digest of Papers., Twenty-First International Symposium.*

Library of Congress. (2017, 12 10). *MAT-File Level 5 File Format*. Retrieved from Libary of Congress: https://www.loc.gov/preservation/digital/formats/fdd/fdd000440.shtml

Lin, T. T., & Siewiorek, D. P. (1990). Error log analysis: statistical modeling and heuristic trend analysis. *IEEE Transactions on Reliability, 39*(4), 419-432.

Luo, F., Yu, T., Kevin, Gupta, A., & Spaggiari, J.-M. (2018, 07 14). *HBase-user] Hbase: Is possible to filter by integer value if the value is saved as string?* Retrieved from grokbase: http://grokbase.com/t/hbase/user/137jrykyzs/hbase-is-possible-to-filter-by-integer-value-if-the-value-is-saved-as-string

Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable real-time systems.* Shelter Island: Manning Publications.

Mathworks. (2018, 06 28). *MAT-File Format.* Retrieved from MathWorks.com: http://www.mathworks.com/help/pdf_doc/matlab/matfile_format.pdf

McNulty, E. (2014, 05 13). *Understanding Big Data: The Seven V's.* Retrieved from Dataconomy: http://dataconomy.com/seven-vs-big-data/

Microsoft. (2017). *Using a Three-Tier Architecture Model.* Retrieved 08 02, 2017, from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/windows/desktop/ms685068%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396

Mittman, B., & Dominick, W. D. (1973). Developing Monitoring Techniques for an On-Line Information Retrieval System. *Information Storage and Retrieval, 9*, 297-307.

MongoDB, Inc. (2018, 03 14). *$lookup (aggregation).* Retrieved from MongoDB: Documentation: https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/

Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NoSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, 16-19.

Neo4j, Inc. (2018, 02 07). *3.2.1 Values and types*. Retrieved from Neo4j: https://neo4j.com/docs/developer-manual/current/cypher/syntax/values/

NOSQL Databases. (2018, 01 26). Retrieved from nosql-database.org: http://nosql-database.org/

Obe, R., & Hsu, L. (2015). *PostgreSQL Up & Running.* Sebastopol: O'Reilly Media, Inc.

Oracle Corporation. (2010). *Storing Transaction Logs in a Database*. Retrieved 08 28, 2017, from Oracle: https://docs.oracle.com/cd/E19501-01/819-3659/gcmam/index.html

Oracle Corporation. (2018, 07 15). *11.8 Data Type Storage Requirements*. Retrieved from MySQL: https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html

PCMag. (2017). *Definition of log.* Retrieved from PCMag: http://www.pcmag.com/encyclopedia/term/46257/log

Peters, T. A. (1993). The history and development of transaction log analysis. *Library Hi Tech, 11*(2), 41-66.

Plattner, H. (2014). The impact of columnar in-memory databases on enterprise system: Implications of eliminating transaction-maintained aggregates. *Proceedings of the VLDB Endowment*, (pp. 1722-1729).

Pritchett, D. (2008). BASE: An ACID alternative. *Queue, 6*(3), pp. 48-55.

Redmond, E., & Wilson, J. R. (2012). *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement.* Dallas: The Pragmatic Bookshelf.

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases: New Opportunities for Connected Data.* Sebastopol: O'Reilly Media, Inc.

Saaty, T. L. (1990). How to make a decision: The Analytic Hierarchy Process. *European Journal of Operational Research*, 9-26.

Saaty, T. L., & Tran, L. T. (2007). On the invalidity of fuzzifying numerical judgments in the Analytic Hierarchy Process. *Mathematical and Computer Modeling*, 962-975.

Sadalage, P. J., & Fowler, M. (2013). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Upper Saddle River: Addison-Wesley.

SAS. (2017). *What is ETL?* Retrieved 07 26, 2017, from SAS Institute, Inc: https://www.sas.com/en_my/insights/data-management/what-is-etl.html

Schneier, B., & Kelsey, J. (1999). Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 159-176.

Schram, A., & Anderson, K. M. (2012). Mysql to NoSQL: Data Modelling Challenges in Supporting Scalability. *Proceedings of the 3rd Annual Conferences on Systems, Programming, and Applications: Software for Humanity*, (pp. 191-202). Tucson, AZ.

ScyllaDB. (2018, 07 15). *Data Maniuplation*. Retrieved from Scylla Documentations: http://docs.scylladb.com/getting-started/dml/

Serra, J. (2012, 03 14). *Data Warehouse vs Data Mart*. Retrieved 07 26, 2017, from JamesSerra.com: http://www.jamesserra.com/archive/2012/05/data-warehouse-vs-data-mart/

Serra, J. (2017, 05 20). *Choosing technologies for a big data solution in the cloud.* Retrieved from James Serra's Blog: https://www.slideshare.net/jamserra/choosing-technologies-for-a-big-data-solution-in-the-cloud

Sharpened Productions. (2017). *Flat File Definition*. Retrieved from Tech Terms: https://techterms.com/definition/flatfile

Simache, C., & Kaâniche, M. (2001). Measurement-based availability analysis of Unix systems in a distributed environment. *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on* (pp. 346-355). IEEE.

Smith, J. M., & Smith, D. C. (1977). Database abstractions: aggregation and generalization. *ACM Transactions on Database Systems (TODS)*, 105-133.

solid IT gmbh. (2016, 11 21). *DB-Engines Ranking of Relational DBMS*. Retrieved from DB-Engines: http://db-engines.com/en/ranking/relational+dbms

Sosnowski, J., Gawkowski, P., & Cabaj, K. (2011). Event and performance logs in system management and evaluation. *Information Systems in Management XIV, Security and Effectiveness of ICT Systems*, 83-93.

Spaggiari, J.-M., & O'Dell, K. (2016). *Architecting HBase Applications: A Guidebook for Successful Development and Design.* Boston: O'Reilly.

Sullivan, D. (2015). *NoSQL for Mere Mortals.* Ann Arbor: Addison-Wesley.

USGS. (2017, 06 20). *USGS Data Management*. Retrieved from USGS: https://www2.usgs.gov/datamanagement/why-dm/lifecycleoverview.php

Vogels, W. (2009). Eventually consistent. *Communications of the ACM, 52*(1), pp. 40-44.

Weisstein, E. W. (2016). *Graph Edge.* Retrieved from MathWorld--A Wolfram Web Resource: http://mathworld.wolfram.com/GraphEdge.html

Weisstein, E. W. (2016, 11 28). *n-tuple.* Retrieved from Mathworld--A Wolfram Web Resource: http://mathworld.wolfram.com/n-Tuple.html

Weisstein, E. W. (2016). *Vertex.* Retrieved from Mathworld--A Wolfram Web Resource: http://mathworld.wolfram.com/Vertex.html

White, T. (2015). *Hadoop: The Definitive Guide.* Sebastopol: O'Reilly Media, Inc.

Yemini, S. A., Kliger, S., Mozes, E., Yemini, Y., & Ohsie, D. (1996). High speed and robust event correlation. *IEEE communications Magazine, 34*(5), pp. 82-90.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
30-08-2018	Doctoral Dissertation	March 2015 – September 2018

4. TITLE AND SUBTITLE

A Methodology for Evaluating Relational and NoSQL Databases for Small-Scale Storage and Retrieval

5a. CONTRACT NUMBER

5b. GRANT NUMBER
18RT0095

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Engle, Ryan D. L., Maj, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENV-DS-18-S-047

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Dr. Erik P. Blasch
875 N. Randolph St., Ste. 325
Arlington, VA 22203
Erik.Blasch.1@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/AFOSR/RTA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Modern systems record large quantities of data capturing time-ordered events, system state information, and behavior. Subsequent analysis enables system status reporting, supports fault investigations, and may provide insight for emerging system trends. Unfortunately, the management of log data requires increasingly efficient and complex storage tools to access, manipulate, and retrieve these records. Historically, database requirements were well-served by relational data models, however modern, non-relational databases, i.e. NoSQL, solutions, initially intended for "big data" distributed system may also provide value for smaller-scale problems such as those required by log data. However, no evaluation method currently exists to adequately compare the capabilities of various solutions for small-scale problems.

This research proposes a methodology to evaluate modern data storage and retrieval systems. While the methodology is intended to be generalizable to many data sources, a commercially-produced unmanned aircraft system served as a use case. The research defined the key characteristics of database technologies and used those characteristics to inform laboratory simulations. Based on those results, twelve evaluation criteria were proposed to compare the database types. The Analytical Hierarchy Process was then used to determine the most suitable database type for the use case. The study results demonstrate the efficacy of the methodology.

15. SUBJECT TERMS

Databases, Relational Databases, NoSQL Databases, Analytical Hierarchy Process, Evaluation Criteria

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Dr. Brent T. Langhals, AFIT/ENV
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	165	(937) 255-3636x7402 Brent.Langhals@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18