12-2021

Electronic Evidence Locker: An Ontology for Electronic Evidence

Daniel Smith
East Tennessee State University

Electronic Evidence Locker: An Ontology for Electronic Evidence

A thesis

presented to

the faculty of the Department of Computing

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Computer and Information Sciences, Information Technology

by

Daniel Smith

December 2021

Dr. Ghaith Husari, Chair

Dr. Brian Bennett

Mr. Stephen Hendrix

Keywords: big data, NoSQL, ontology

ABSTRACT

Electronic Evidence Locker: An Ontology for Electronic Evidence

by

Daniel Smith

With the growth of technology, cases experience overwhelming amounts of electronic evidence that need to be stored and shared. Little research in developing methods for sharing electronic evidence between agencies currently exists. One problem is the number of records that must be stored. Relational database solutions face size limitations when storing large amounts of data. Another problem involves storing evidence where each instance has unique attributes. The Electronic Evidence Locker (EEL) was proposed and developed to address these problems. The EEL was built using a NoSQL database for storage and a C# website for users to query stored data. Baseline results collected measure the growth in required machine resources and time when adding additional search criteria and larger data sets. These results show search time is impacted more by the search direction and number of layers travelled than by the addition of search conditions to the query.

DEDICATION

I dedicate this thesis to my amazing wife, Jenn Smith. Without her unwavering support and encouragement, I do not think I would have finished this. As we say to one another, “Always”.

I also want to dedicate this to my family, who always held me accountable and pushed me to do my best.

Last, but not least, I would like to dedicate this to God. Through God, all things are truly possible.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ghaith Husari, my committee chair, for going above and beyond to assist me with this thesis. I am forever grateful for the support he provided through countless conversations, emails, reviews, and discussions.

I also want to thank Dr. Brian Bennett and Stephen Hendrix for their contributions as committee members. Throughout this thesis, there were many times I would ask them questions and they always helped me find my way.

I also want to thank Dr. Michael Lehrfeld for his help in establishing what this thesis would accomplish.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1. Introduction

The ability to share evidence requires a new storage model for the large quantity of electronically stored information (ESI) recovered through electronic discovery (e-discovery). E-discovery is the process of retrieving and storing ESI during criminal and civil cases. E-discovery can involve devices and networks owned by either an individual or a company. Since e-discovery can retrieve millions of records, a storage model that aids in digitally sharing ESI benefits law enforcement agencies and legal personnel. The electronic sharing of ESI saves law enforcement agencies and legal staff time and money. Furthermore, a new storage model addresses the rapid growth of retrieved ESI.

Currently, law enforcement agencies do not have a way to share ESI with legal personnel digitally. The current method for sharing ESI requires legal personnel to travel to the agency physically storing the ESI. Once on-site, legal personnel can either review the ESI or transfer it to an electronic medium, whether a disk or portable hard drive, for review at a different location. However, the copying of certain content, such as content containing child pornography, is legally restricted. Since current sharing methods require travel and possibly physical items, there is a direct cost associated with the current method. The ability to view the electronic evidence through a web portal reduces expenses for legal staff related to travel and hardware.

The first step to digitally sharing ESI is to study how to store data retrieved through e-discovery. With individual devices containing large amounts of data, ESI storage methods impact the ability to search for specific content. The method for storing ESI must account for the contents of the ESI and meta-data of the device that housed the data. Examples of such meta-data include the operating system and the operating system's version number. However, the ability to handle large amounts of varied data is not a unique problem to law enforcement agencies.

Businesses and scientific communities addressed the problem using tools built specifically to handle large quantities of mixed data received from various sensors, devices, and inputs. This type of problem is known as a big data problem.

The next step in digitally sharing ESI is to develop a model that stores the information retrieved through e-discovery using solutions to big data problems. This thesis describes the development of the Electronic Evidence Locker (EEL) tool using big data techniques. The remainder of the thesis is organized as follows. Chapter 2 discusses how organizations store big data, a review of the NoSQL solutions used, and how ontologies assist with modeling data. Chapter 3 presents how both the ontology and the EEL tool were built. Chapter 4 describes how the sample data was generated and seeded, how the testing procedures were created, and the results observed from the test cases. In Chapter 5, the thesis concludes with what performances were observed and where future work with the EEL could go.

Chapter 2. Literature Review

2.1 Storing Big Data

Designing new solutions and frameworks must occur continuously to address the growth of technology. One example of technology growth is the size and complexity of the data that users store and analyze. Companies and other organizations use the term big data to describe large quantities of data. In addition to size, several other properties describe big data: variety, velocity, variability, complexity, and value.

One property associated with big data is variety. Variety represents that "data being produced is not of a single category or type" [1]. Big data can consist of emails, documents, sales patterns, and sensor readings. The variety of data leads to different data structures that result in query processing difficulties in current analytic engines. Another property of big data is velocity. Velocity refers to the speed at which data comes and goes to different sources. Variability refers to how data flows and whether the requested data is consistent or requested only at specific times. Big data contains a complexity property as well. The complexity property pertains to managing data that comes from various sources and transforming it to a common standard. A final property associated with big data is value. The value property addresses the ability for users to "acquire business trends and change their strategy accordingly" [1]. Organizations take advantage of these big data properties to aid with organizational operations.

Different organizational groups benefit from big data in different ways. For example, an Information Technology (IT) group benefits from big data when storing and maintaining logs for running services. The structure of the logs generated by services can be different, complicating large-scale analyses for errors. Since big data allows for varied data types, IT can store the logs in a particular location and analyze the variety of logs using a general format. A second example where an organizational group benefits from big data is the financial group using big data to create financial models. Finance can use these data models to “calculate risk so that it falls under their acceptable thresholds” [1]. A third example is an ability for manufacturing groups to use big data to take advantage of sensor readings. Manufacturing faces the problem of not having the infrastructure in place to store and analyze the readings generated by various sensors in the plants. However, big data allows for the storage of sensor readings, which can allow the manufacturing group to increase profit and manufacturing efficiency by analyzing the variety of readings. A final example of an organizational group benefitting from big data is the manner in which marketing uses big data to evaluate customer feedback through social media. Retrieving and storing customer feedback from various social media platforms allows marketing to “modify decisions and get more value out of their business” [1]. The advantages gained by different organizational groups are not without problems, however.

All new technology has challenges and issues. The first, and likely most important, issue with big data involves privacy and security. These concerns include who can access the data and fears regarding discovery of personal secrets when combining personal information with other data sets. Another privacy concern is whether individuals are aware that companies collect personal information in the first place.

Because of the volume of data, an organization’s ability to store and analyze big data becomes another issue. Since big data could contain petabytes of data, businesses need to plan their IT infrastructures accordingly. Businesses could either store big data on-site, paying for hardware, or transfer the data to the cloud, paying for services. When a business moves big data to the cloud, however, they cannot make real-time decisions because uploading big data takes time. Another challenge involves the analysis of the data. There are questions of "what needs to be stored, what needs to be analyzed, and what data points are important" [1] that the business must answer. Staff skillsets and technical challenges, including fault tolerance and scalability, can become problematic as well. While software cannot address issues about business processes and how businesses collect data, software can address technical issues associated with big data.

2.2 SQL vs. NoSQL

Organizations traditionally store structured data in relational databases known as SQL databases. However, the variety of structures within big data becomes challenging when attempting to use an SQL database. Furthermore, the ability to "process such vast amounts of data requires speed, flexible schemas, and distributed databases" [2], which SQL databases struggle to offer. To address the concern of storing big data, organizations have turned to NoSQL databases.

NoSQL databases have several variations: key-value pairs, column-family databases, and document-oriented databases [2]. Key-value pair databases store data as a pair of related entities, with primary identifying entity being assigned as the key and the content is the value. This allows "programs to retrieve data by keys, which are essentially names, or identifiers, that point to some stored value" [3]. Column-family databases group data into columns and then store those columns in a super column. The super column "groups a number of related columns and can be accessed as a single unit" [2]. Examples of column-family databases include Cassandra and Hypertable. Document-oriented databases allow "for the storage of objects in serialized forms as well as simple values" [2]. The following applications are examples of document-oriented databases: MongoDB, CouchDB, RavenDB, and Couchbase. While NoSQL databases offer improvements in data replication and data storage, the question of whether NoSQL databases can outperform SQL databases in fundamental operations remains.

For Li and Manoharan in [2], the first aspect to decide whether NoSQL databases performed fundamental operations better than SQL databases was selecting databases to compare. Microsoft SQL Express was compared to several NoSQL databases because of the varied nature of NoSQL: Cassandra, Couchbase, CouchDB, Hypertable, and MongoDB. The next step was selecting operations to evaluate. The chosen operations included instantiating database buckets, reading data keys, writing data to the database, deleting data, and retrieving all keys from the database. The first test evaluated the "time taken to instantiate a database bucket, using the average over five runs" [2]. The operations for evaluating the reading, writing, deleting, and retrieving the number of keys occurred a specific number of times. These operations took place the following number of times: 10, 50, 100, 1,000, 10,000, and 100,000. Like the instantiation test, each of the reading, writing, deleting, and retrieving tests occurred five times and showed the averages of the runs.

The instantiation test results in [2] showed that all the NoSQL databases initialized faster than the SQL database. However, while SQL took over 1,600 milliseconds (ms) to instantiate, the NoSQL databases ranged from RavenDB taking 200ms to Couchbase, which took over 1,500ms to instantiate [2]. This trend of wildly ranging values in the NoSQL databases occurs throughout all the tests.

The results for the read test in [2] show that only Couchbase and MongoDB perform better than SQL Express. While Couchbase outperforms MongoDB in total time of reading, MongoDB outperforms Couchbase up to 100 operations [2]. Once past 100 operations, Couchbase surpasses MongoDB's abilities to perform reads [2]. The values range from

Couchbase performing 100,000 reads in 7,244ms to RavenDB performing the same number of read operations in 426,505ms. Then compare the range of NoSQL results to SQL Express, which performs 100,000 reads in 17,214ms. The writing test results share similar results.

The writing test in [2] showed that four of the six NoSQL databases performed better than SQL. The four that performed faster than SQL Express were "Couchbase, MongoDB, Cassandra, and Hypertable" [2]. Like the reading test, MongoDB performed faster than Couchbase at a lower number of operations. However, Couchbase performed better than MongoDB at even 100 operations. Note that while Cassandra and Hypertable are faster than SQL Express, the two NoSQL databases are only faster once there are more than 100 operations. To see the wide performance range between the NoSQL databases, Couchbase performs 100,000 writes in 8,492ms while CouchDB performs the same number of writes in 932,038ms [2].

The results of the delete test in [2] echo the results of the read test, with the only NoSQL databases to outperform SQL Express being Couchbase and MongoDB. Like the writing test, MongoDB only outperformed Couchbase up to 50 operations, with Couchbase performing the delete operations faster at 100 and more operations. The results for the NoSQL databases ranged from Couchbase performing 100,000 deletes in 7,634ms to RavenDB performing the same number of deletes in 799,409ms [2]. In comparison, SQL Express performed the 100,000 deletes in 32,741ms. The results of the final test are different from any of the other tests.

The final test, the fetching of all keys, results in none of the NoSQL databases performing better than SQL Express. One difference between this test and the others is that Couchbase does not have "an API to support fetching all the keys, and thus was excluded from this test" [2]. The results showed SQL Express fetching 100,000 keys in 76ms. The closest

NoSQL database was Hypertable, which retrieved 100,000 keys in 159ms. The furthest NoSQL database, CouchDB, took 9,512ms.

These experiments found that not all NoSQL databases can perform fundamental operations faster than an SQL database. Testing demonstrated that Couchbase and MongoDB were the only two NoSQL databases to outperform the representative SQL database consistently. It is important to note that the testing performed "did not test the databases for more complex operations" [2]. Despite the lack of complex testing, the wide range of performance capabilities prove that NoSQL databases are not all created equally. While not all NoSQL databases proved to be acceptable, MongoDB consistently outperformed its SQL counterpart.

2.3 MongoDB vs. MS-SQL

Wu, Huang, and Lee further show the advantage of using MongoDB over a relational database in [4] where MongoDB reads and writes faster than Microsoft SQL (MS-SQL). The data source for the tests came from the Taiwan Water Company website. The first part of the experiment evaluated the reading and writing operations individually in MongoDB and MS-SQL. The operations ran in the following increments: 10,000, 50,000, 100,000, 500,000, and 1,000,000 operations. The experiment then assessed each operation in a single-threaded instance and a multi-threaded instance. This series of assessments evaluated the operations over the course of a million runs. The final aspect of the experiment evaluated how much of a role indexing played when searching for data in either database management system. Across all tests, the results show that MongoDB outperforms MS-SQL. The single-threaded reading and writing tests from [4] show that MongoDB performs close to ten times better than MS-SQL.

In the multi-threaded writing test, MongoDB still outperformed MS-SQL but not to the same extent. In the multi-threaded writing test, MongoDB performed twice as fast. The multi-threaded reading test echoes the results of the single-threaded reading test, but because the researchers capped the Y-axis on the MS-SQL chart at 100,000ms the full performance difference is unknown. The final evaluation showed that searching for indexed data is more efficient than non-indexed data in MongoDB and MS-SQL. While the "performance of searching is naturally better" [4] with indexed data, MongoDB's indexed searching was almost ten times better and MS-SQL's indexed searching only improved under three times better than non-indexed searches.

While relational databases work well with structured data, the inability for an RDBMS to work with large quantities of data limits its uses. Furthermore, for MongoDB to perform simple operations more efficiently further limits the utility of MS-SQL for big data operations. However, MS-SQL is not the only RDBMS available.

2.4 MongoDB vs. MySQL

Nyati, Pawar, and Ingle show in [5] that MongoDB performs insert and searching operations faster than MySQL. The insert test format involves inserting 500,000 records, where each record instance contained 28 columns, into the database. The searching test consisted of searching the 500,000 previously inserted records.

The results of the insertion test had "MySQL inserting 500,000 records in 1,606 seconds while MongoDB took 17.8 seconds" [5] to complete the same task. This test shows that MySQL took 89 times longer than MongoDB. While MongoDB performed the search test faster than MySQL, it was not to the same degree of improvement. When searching by columns with an index, MongoDB performed just under 24 times better than MySQL. The performance advantage decreased further when searching for columns without an index with MongoDB performing just 6.5 times better than MySQL. While there is data for searching by primary key for the MySQL database, there is no such data provided for MongoDB to compare against. While MongoDB performed far better than MySQL, the insert and search tests only ran on a single thread. To evaluate MongoDB further, tests evaluated how MongoDB performed over a large data set using multiple threads.

To see how MongoDB handled large data sets, researchers Nyati, Pawar, and Ingle [5] created tests that used a defined number of calls and a set number of threads to run on a database containing 50 million records within it. The number of calls is “a repeated call of the same query with a different value” [5] to the database. For this test, the search query is the call used. The number of calls chosen for the tests are the following: 2,000, 5,000, 7,000, and 25,000 calls. The number of threads used for the tests fall in the following list: 1, 3, 5, 6, and 20 threads. To prevent abnormal results skewing data, these performance tests took place a various number of times and the average of those results chosen.

The first result of the large dataset testing showed that even with a higher number of threads, searching a cluster is faster than performing a search on a single machine. The lowest time to perform 5,000 calls on a single thread on a single machine was 923ms where the same number of calls performed on a cluster using 20 threads took 687ms [5].

Table 1. Summary of All Searches from [5]

Call	Thread	Time required (ms)
2000	1	188
2000	3	221
2000	5	260
5000	6	331
5000	20	687

7000	20	3287
25000	1	5997

The results, shown in Table 1 sourced from [5], show that introducing more threads into the search increases the time required to perform the search. However, when the number of calls is greater than 5,000, there is a rapid growth in time required to complete the search. To mitigate this performance degradation, the “configuration and sharding in MongoDB must have a higher importance to get higher throughput” [5]. The results of the MySQL and MongoDB comparison once again show that MongoDB outperforms relational databases in simple operations. MongoDB performs 89 times better with insertion operations and at least 6.5 times better with searching. Finally, the large data set testing finds the point at which performance degradation begins due to the large number of calls. With this knowledge, the configuration and sharding of MongoDB can be tweaked to achieve better results.

2.5 Ontology

An ontology is a formal specification and representation of the categories, properties, concepts, and relations between them [6]. It consists of concepts that are relevant to the domain of interest. Each of these concepts may have properties (data). The relations between the concepts provide the cornerstone for modeling high quality, linkable, and coherent data. This makes the ontologies highly effective frameworks for representing shareable and reusable knowledge in the domain of interest. Ontology can be broadly categorized into three levels: (1) top-level, (2) middle- (utility) level, and (3) bottom-level [6].

- Top-level ontology. Higher concepts that provide common knowledge that is not domain specific.

- Utility ontology. Contains concepts are less abstract than the top-level. Such concepts encompass concepts with more details that are usually found in classes (or categories) of concrete concepts.
- Bottom-level ontology. This level contains concrete instances that are highly domain specific. Instances in this level may extend the concepts in the Utility level. Figure 1 illustrates the different levels of an ontology.

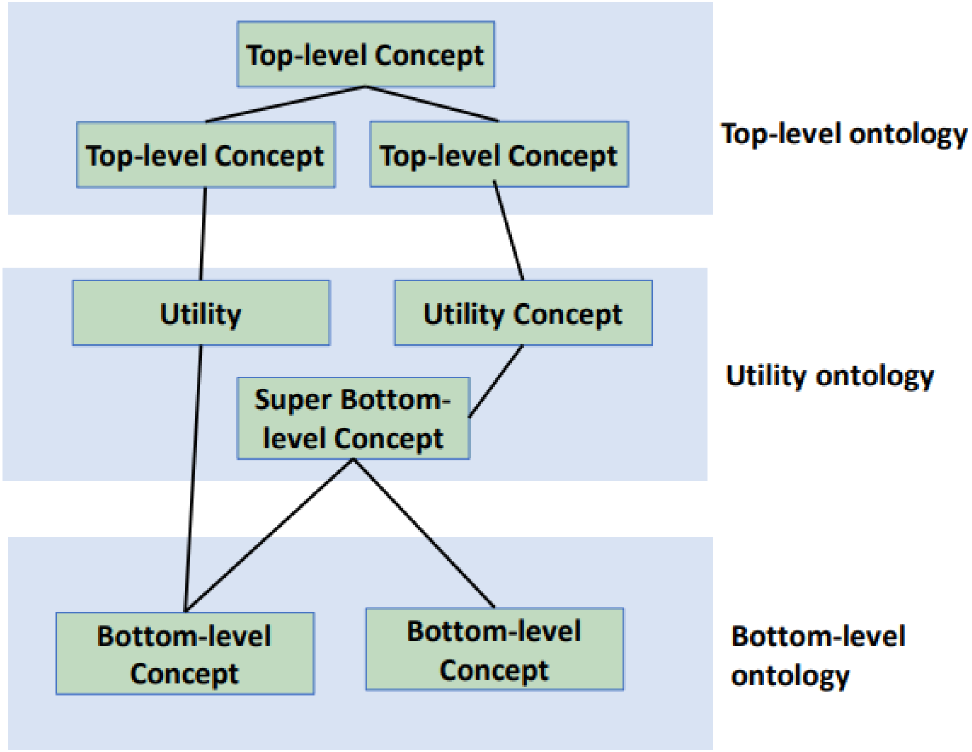


Figure 1. The Different Levels of an Ontology

For this thesis, an ontology helps define how case and evidence will be modeled into objects to be stored in the NoSQL database. While NoSQL allows for objects to be dynamically structured, providing a structured object will assist in building features that edit and retrieve data.

Chapter 3. Design and Implementation

3.1 Approach Design and Architecture

The development of the EEL ontology began with deciding what elements were required to be searchable. With the large domain of evidence available, this project's scope limited itself to storing and searching phone calls and text messages retrieved through phone dumps. However, after reviewing phone dumps generated from the Cellebrite software, the focus shifted from specialized phone dumps to generalized data dumps, leaving the building of specific data dump extractions to later. After shifting to generalized data dumps, the ontology shown in Figure 2 was designed to capture the following elements: case, crime, evidence, and content.

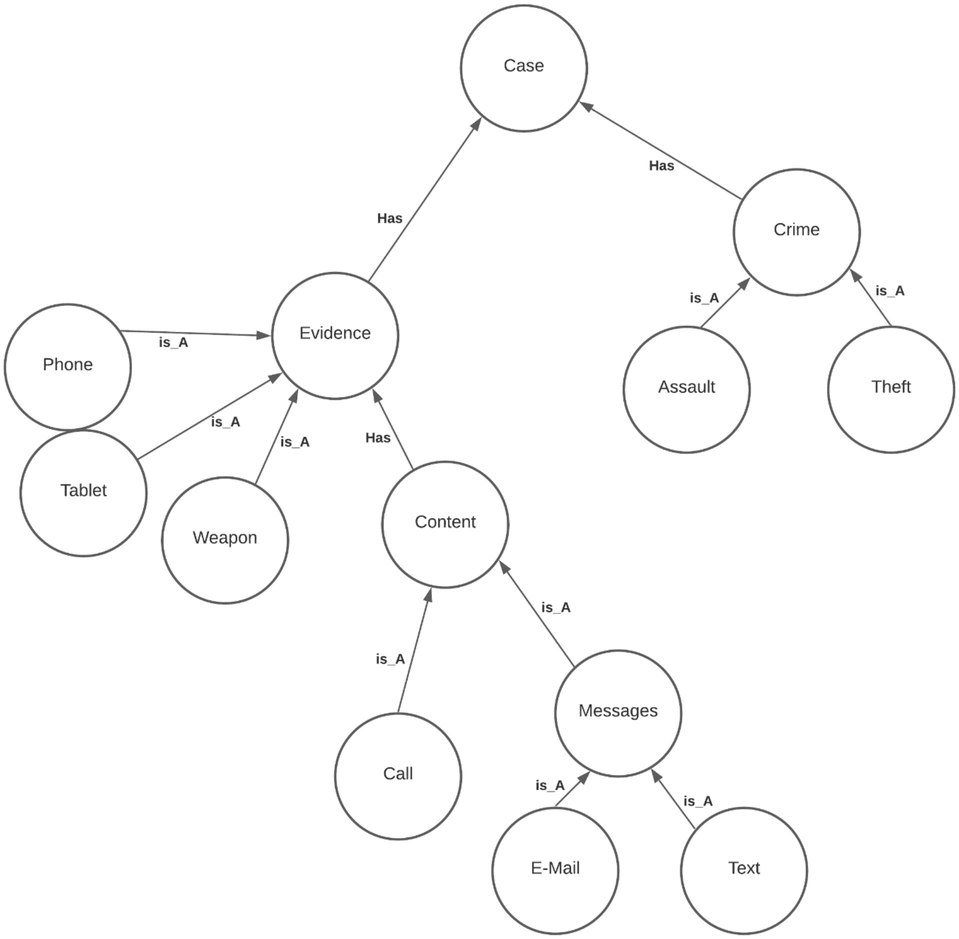


Figure 2. Electronic Evidence Locker Ontology

3.1.1 Case

A case is the key class of the ontology. A case has attributes that give general information about the case. Example attributes include the case number used by law enforcement organizations and the lead agent of the case. Based on the case's type, type-specific attributes could be attached as well.

3.1.2 Crime

A crime is a type of law violation associated with a case. When a case first begins, there may not be any defined crimes linked to it. Once a crime is confirmed, it is then linked to the associated case.

3.1.3 Evidence

Evidence is an item that contains something relevant to the case. Evidence could be physical items, e.g., cellphones, tablets, hard drives, or weapons. Evidence can also include digital items, like data dumps of emails and documents. Like the case class, some attributes give general information about the piece of evidence, e.g., the location of where the evidence was found or the individual who examined the evidence. Based on the type of evidence, other attributes can be assigned as well. For example, instancing the evidence as a phone will attach attributes like manufacturer and MEID to the class. A case can have zero to many pieces of evidence.

3.1.4 Content

Content is defined as something that resides within a piece of evidence. An example of content would be a text message or a call stored on a cellphone. Like the case and evidence classes, attributes are assigned based on the content type. For example, a content instanced as a text message would have an attribute for the body of a message, while a content instanced as a call would have an attribute for the call's duration.

3.2 Implementation

Testing the effectiveness of the EEL ontology required building a tool that used a NoSQL database and a website for users to interface with. Previous research shows the higher performance of NoSQL databases compared to alternative solutions. Considerations for the website included ability to access the service and ease of hosting.

3.2.1 Storage

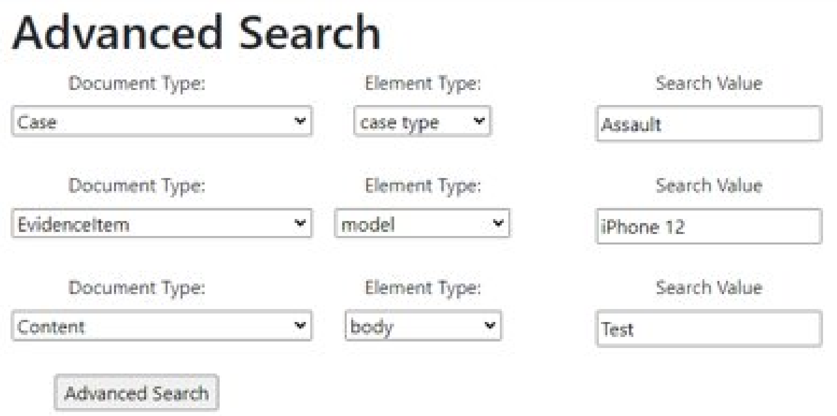
The NoSQL database selected was MongoDB v.4.4.6. One advantage that MongoDB provides is that MongoDB can store an unlimited number of documents [7] within a collection, which avoids space limitations presented by RDBMS solutions. Another advantage of NoSQL databases involves the faster read and write speeds shown in papers [2] [4] [5].

The database's configuration is that a single collection stores all the documents. By using a single collection, users can perform predictive analysis to draw connections between stored content, evidence, and cases. This collection stores each ontology level in its own document. Each document consists of the following fields at a minimum: id, parent id, and document type. The id is a unique identifier for the document. The parent id field is any object that the current document is associated with. The document type field is evaluated to determine what the document's level is within the ontology.

3.2.2 Website

The website was built using the C# language and the .NET Core 3.1 framework. Two factors led to the use of this framework. The first is the number of established libraries developed for .NET Core reduced the amount of code created. The second is because the .NET

Core framework is operating system agnostic. Therefore, .NET Core allows for housing the system on almost any server available, whether an on-premises Windows 2019 server or a Linux-based server hosted by Amazon Web Services. The website created consists of the following two features: a way to build the documents stored in the MongoDB server and a way for users to search the stored documents with a graphical interface. Figure 3 shows the advanced search feature built into the website.



Advanced Search

Document Type:	Element Type:	Search Value
Case	case type	Assault
EvidenceItem	model	iPhone 12
Content	body	Test

Figure 3. Example of Advanced Search - EEL Front End

3.2.2.1 Model

While the ontology only has three levels in the hierarchy, each level is only required to hold an id, parent id, and document type field. The conversion of the ontology into an entity relationship model is shown in Figure 4. To allow for different versions of each level, a base class exists for each document level that contains the mandatory fields. Specialized document types inherit from these base classes. When new documents are created, they are created as a specialized document instance and then converted to a binary JSON (BSON) document and saved into the MongoDB collection. BSON, based on the JavaScript Object Notation (JSON) data format, extends upon JSON functionality and “encodes type and length information, which allows it to be parsed much more quickly” [8]. There are two reasons why each level is created as a unique document. First, there is no guarantee that the size of the documents will stay small. Second, the manner in which data is searched is uncertain. MongoDB states “if you often only need to retrieve a subset of documents within the group, then ‘rolling-up’ the documents may not provide better performance” [9].

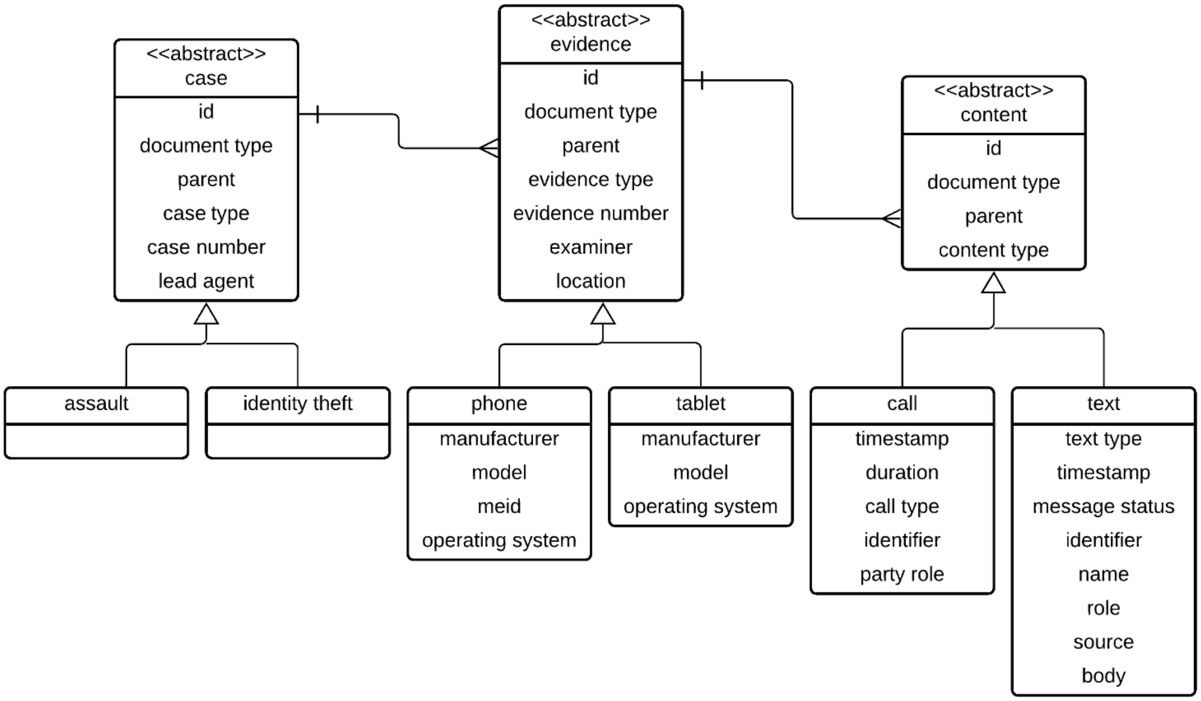


Figure 4. Electronic Evidence Locker Entity Relationship Model

One advantage of using C# with this type of construction is that, since it is strongly-typed and a compiled language, errors associated with known data issues will be made aware before deploying a class. A disadvantage, however, is that when new document types are added to the MongoDB collection, the models and views associated with the new document type must be defined. This disadvantage requires an upfront resource cost to create as many classes at the beginning as possible.

3.2.2.2 Search

The search feature begins by creating a MongoDB aggregation pipeline. A pipeline is a series of steps that are performed in sequential order and can be used to transform and filter collections of data. The first step in the pipeline locates all documents that meet the first criteria. An assumption is that the documents returned by the search will match the document type of the first criteria.

The search method then determines what the current document type level is. It then evaluates the document type levels for the second and third criteria, if they are present. If the difference between the primary criteria level and the secondary criteria level or the primary criteria level and the third criteria level are greater than one, the search assumes a deep search is needed.

The search will then insert graph lookup methods provided by MongoDB to link all needed parent and child objects to the current object into the pipeline. If both parent and child objects are included in the search, a step to merge the two arrays created from the graph lookups into a single array will be added to the pipeline. The final step of the pipeline is to perform a match based on the criteria and return the results to the user.

The following example describes how the search is performed. This search would look for all evidence with an evidence type of phone linked to a case where the case type is assault. This sample search consists of the following two search criteria: the first scenario is looking for all evidence that the *evidence type* is a *phone* and the second scenario is looking for cases where the *case type* is *assault*.

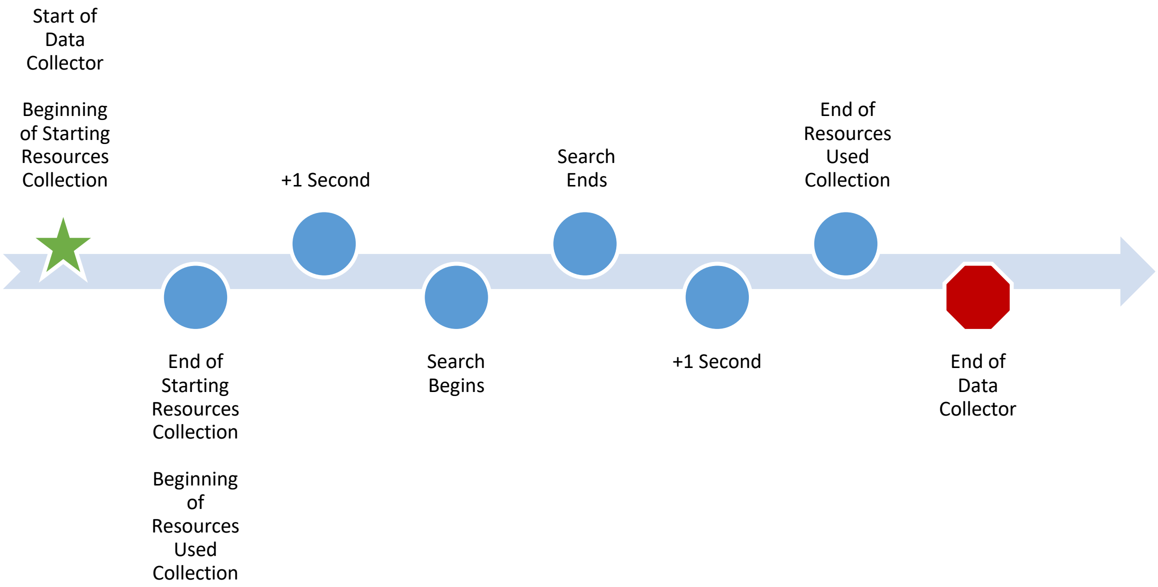
The search starts by finding all documents that contain a field of *evidence type,* and that field has a value of *phone*. Since the search has a second criterion, the search method compares the document type of the first and second criteria. It determines that the second criterion is not of the same level as the first criterion. Because the case document level is higher than the document level for evidence, the search includes parent objects for the second criterion. After joining the parent objects, the search then looks at the documents within the parent objects for a field of *case type* and where the value is *assault*.

Chapter 4. Evaluation

To evaluate the effectiveness of the EEL ontology, six scenarios were devised to assess the accuracy and required time and space resources needed during the search process. The resources considered included the amount of Random-Access Memory (RAM) used and the percentage of the CPU used during the search processes. The machine used for this evaluation had the following specifications:

- CPU: Intel i7-4790k, base speed 4.00GHz
- Memory: 32GB DDR3, speed 2400MHz
- Storage: 1TB SATA SSD
- Operating System: Microsoft Windows 10
- MongoDB Server: 4.4.6
- .NET Core Framework: 3.1
- Visual Studio: Microsoft Visual Studio Community 2019
- Background processes: MongoDB Server, standard Windows 10 services

The tool Performance Monitor, included with Windows 10, captured the amount of RAM used and the CPU usage. The value used to calculate memory use was the Memory – Available Bytes counter. When determining the amount of RAM used, the minimum value of the range is used. To determine the CPU usage, the Processor Information (Total) - % Processor Utility counter was used. When calculating the amount of CPU usage, the maximum value of the range was used. These values use the respective minimum and maximum values to show a worst-case scenario for the resource usage.


Start of Data Collector

Beginning of Starting Resources Collection

End of Starting Resources Collection

Beginning of Resources Used Collection

+1 Second

Search Begins

Search Ends

+1 Second

End of Resources Used Collection

End of Data Collector

Figure 5. Data Collection Timeline

To calculate the starting resources used by the web application, the values used the range from when the data collector job ran up to one second before the job was reported to have run, as shown in Figure 5. The reason for gathering one second less than the reported start time was to account for when the test actually started, since the original time for gathering the results did not account for fractions of a second.

Equation 1. Starting Usage Formula

$$(Start\ of\ Data\ Collector):(Start\ of\ Test\ Run-1\ Second) \qquad (1)$$

To calculate the resources used during the search, the range from when the job started minus one second to when the job ended plus one second were used. As stated with the starting resource values, the start begins one second before the reported start and the end value ends one second after the reported test run. This configuration is important to accommodate any instabilities (or spiking) in the machine resources caused by executing search queries. In addition, this time range helps to guarantee capturing the range of values during the test run.

Equation 2. Resource Used Formula

$$(Start\ of\ Test\ Run - 1\ Second):(End\ of\ Test\ Run + 1\ Second) \tag{2}$$

To calculate the amount of memory used during the search, the amount of memory available from the resources used was subtracted from the starting amount of memory used. To calculate the amount of CPU used during the search, the amount of CPU used at the start of data collection was subtracted from the CPU used during the search.

One thing to note is that the values collected show the worst-case scenario for how the ontology performs while being searched. The purpose of this thesis was to create a baseline value for how the data built with ontology handles search performance.

4.1 Testing Procedures

To run a test, the first step was to start the debugging process within Visual Studio 2019. The debugging process ran for at least ten seconds before the Data Collector job defined in Performance Monitor started. This lead time allowed for resource usage due to Visual Studio startup to be omitted from the report. Once the initial time passed, the Data Collector job was started and ran for a minimum of five seconds to obtain the starting values for the amount of RAM and CPU used. After the starting values were collected, the test case was then performed. Once the test finished, the data collector ran for at least five more seconds. Each test instance was run five times, and the results averaged. The conditions for each test are defined in Table 2.

Table 2. Test Scenarios

Search Complexity	Conditions
1 - Simple	Evidence.Operating System = KaliOS
2 – Intermediate – Top Down	Case.Case Agent = Khaleesi AND Evidence.Operating System = KaliOS
3 – Intermediate – Bottom Up	Evidence.Operating System = KaliOS AND Case.Case Agent = Khaleesi
4 – Intermediate – Top Down Full	Case.Case Agent = Tyrion AND Content.Body = Drunk
5 – Intermediate – Bottom Up Full	Content.Body = Drunk AND Case.Case Agent = Tyrion
6 - Complex	(Evidence.Operating System = KaliOS AND Case.Case Agent = Khaleesi) OR (Evidence.Operating System = KaliOS AND Content.Body = Ragnarok)

4.1.1 Simple Test (Test 1)

The simple test searched all fields within the stored documents for “Operating System” and looked for a value of “KaliOS”. This search was an exhaustive search that looked at all documents to see if a field of “Operating System” exists. This query was deemed simple because it only had a single condition evaluated, and there was no need to search through linked objects. This test establishes a baseline value for searching through a large collection of documents for a single field and value.

4.1.2 Intermediate Tests (Tests 2-5)

The intermediate tests retrieved results that met the combination of two search queries. Since multiple conditions could be evaluated, tests were devised to accommodate different situations. Tests 2 and 3 evaluated searches where the second search query was one level below or above, while tests 4 and 5 evaluated searches where the second search query was two levels away from the first.

Furthermore, a different condition evaluated is the direction in which the search took place. Tests 2 and 4 evaluated a search that began with parent-level documents and searched down to the attached children documents. Tests 3 and 5 began with child-level documents and searched up to any attached parent documents.

4.1.3 Complex Test (Test 6)

The complex test consisted of several statements that must be true to produce results. The complex query retrieved results that met either the first search query and the second query or results that met the first search query and the third search query. To add to the complexity of the search, neither the second nor the third search queries existed on the same level as the first search query. This test's purpose showed the results of a compound search and how adding additional queries could increase the resources required to run. Since the intermediate test cases addressed a variety of scenarios, the complex test combined a single-level, top-down scenario and a single-level, bottom-up scenario and recorded the results of the search.

4.2 Seed Data Generation

4.2.1 General Data Generation

The test Mongo collections were built automatically with a method that took inputs for the number of cases and whether the data linked to them would be generated statically or dynamically. For these tests, the data was generated with the static option. For each case created, ten pieces of evidence were attached to the case. For each piece of evidence created, ten pieces of content were attached. Therefore, for each case that was generated, 111 documents were created. To give the data in the collection a semblance of realism, lists of values were created for each field type, and each field was populated with a random option from the respective list.

Table 3. Testing Data Sets

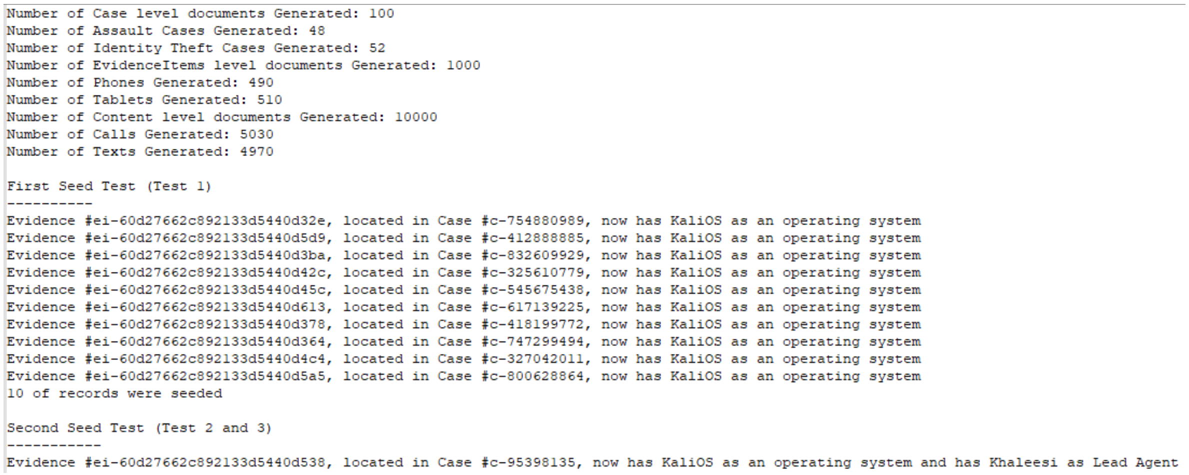
Data Set	# Of Cases	# Of Documents	# Of Searchable Fields
XS – Extra-Small	100	11100	116000
S - Small	500	55500	580730
M - Medium	1000	111000	1161016
L - Large	5000	555000	5804742
XL – Extra-Large	10000	1110000	11610158

For scalability testing, five data sets were created. The data sets shown in Table 3 show that the number of cases range from extra-small with 100 cases to extra-large with 10,000 cases, indicating a range of 11,100 to 1,110,000 total documents. Since this thesis was intended for handling instances of evidence in a big data manner, the testing results from the medium data sets and larger receive the most focus.

4.2.2 Seeding Test Data

To seed the test data, a method was created that selected ten percent of the data set's cases became the number of test cases. If a data set had a hundred cases, then ten records of the designed test type would be seeded in the data set. The documents that were changed into seeded test data were randomly selected by the seeding method and updated to have the respective testing values.

Another consideration for seeding test data was to only seed data in documents that could have the appropriate field, such as making sure that a body field was not added to a phone call document. To simplify reporting, the linked case for any document used in a test scenario was removed from the test seeding process. Doing so also prevented any document linked to the case from being used for two separate test scenarios, which could have skewed the reported results. While generating the test cases, the modified records were noted in the seed generation documents, as shown in Figure 6.

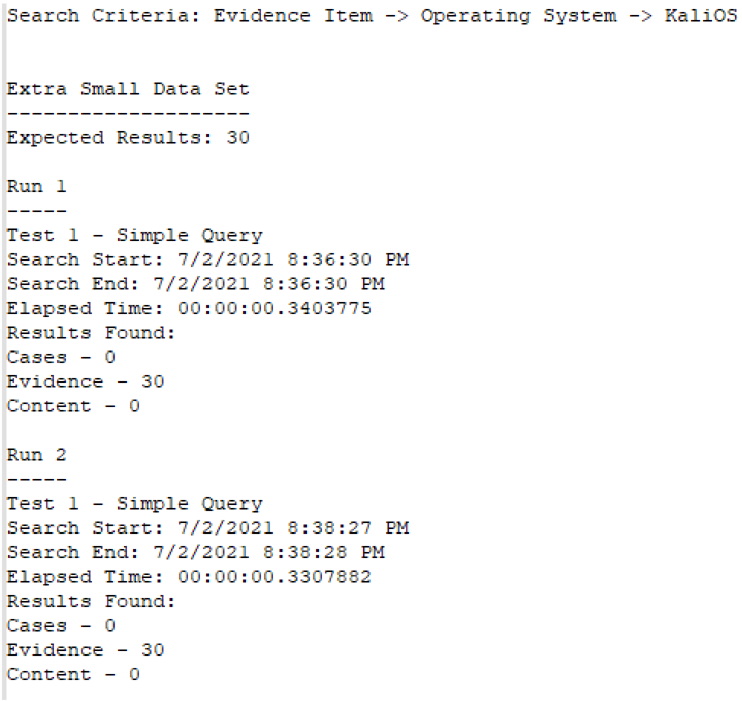


```
Number of Case level documents Generated: 100
Number of Assault Cases Generated: 48
Number of Identity Theft Cases Generated: 52
Number of EvidenceItems level documents Generated: 1000
Number of Phones Generated: 490
Number of Tablets Generated: 510
Number of Content level documents Generated: 10000
Number of Calls Generated: 5030
Number of Texts Generated: 4970

First Seed Test (Test 1)
----------
Evidence #ei-60d27662c892133d5440d32e, located in Case #c-754880989, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d5d9, located in Case #c-412888885, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d3ba, located in Case #c-832609929, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d42c, located in Case #c-325610779, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d45c, located in Case #c-545675438, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d613, located in Case #c-617139225, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d378, located in Case #c-418199772, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d364, located in Case #c-747299494, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d4c4, located in Case #c-327042011, now has KaliOS as an operating system
Evidence #ei-60d27662c892133d5440d5a5, located in Case #c-800628864, now has KaliOS as an operating system
10 of records were seeded

Second Seed Test (Test 2 and 3)
-----------
Evidence #ei-60d27662c892133d5440d538, located in Case #c-95398135, now has KaliOS as an operating system and has Khaleesi as Lead Agent
```

Figure 6. Seeding Results for Extra-Small Data Set

4.3 Accuracy Tests and Measures

To test the accuracy of the search, the number of records found by the search were compared against the number of noted test cases built. The number of noted test cases were combined to create an expected results number that was noted before the tests began. The expected results number was then compared against the number retrieved by the search, as shown in Figure 7. For every test run, the search found 100% of the records that were seeded.



```
Search Criteria: Evidence Item -> Operating System -> KaliOS


Extra Small Data Set
--------------------
Expected Results: 30

Run 1
-----
Test 1 - Simple Query
Search Start: 7/2/2021 8:36:30 PM
Search End: 7/2/2021 8:36:30 PM
Elapsed Time: 00:00:00.3403775
Results Found:
Cases - 0
Evidence - 30
Content - 0

Run 2
-----
Test 1 - Simple Query
Search Start: 7/2/2021 8:38:27 PM
Search End: 7/2/2021 8:38:28 PM
Elapsed Time: 00:00:00.3307882
Results Found:
Cases - 0
Evidence - 30
Content - 0
```

Figure 7. Retrieval Search Results for Simple Query

4.4 Results

For thorough testing, the data sets range from 100 total cases to 10,000 total cases. However, after reviewing the testing results, the extra-small (100 cases) and small (500 cases) data sets exhibited unstable behaviors. For the following results, the patterns established come from working with the medium (1000 cases) to extra-large (10,000 cases) data sets.

4.4.1 Baseline Results from Simple Test

The charts in Figure 8, Figure 9, and Figure 10 establish baseline results for the time taken, memory used, and processor utilization percentage used for running a simple search (Test 1) across the five data sets. These figures indicate that an increase in the size of the data set results in a near-constant or weak-linear increase in the amount of the resources used.

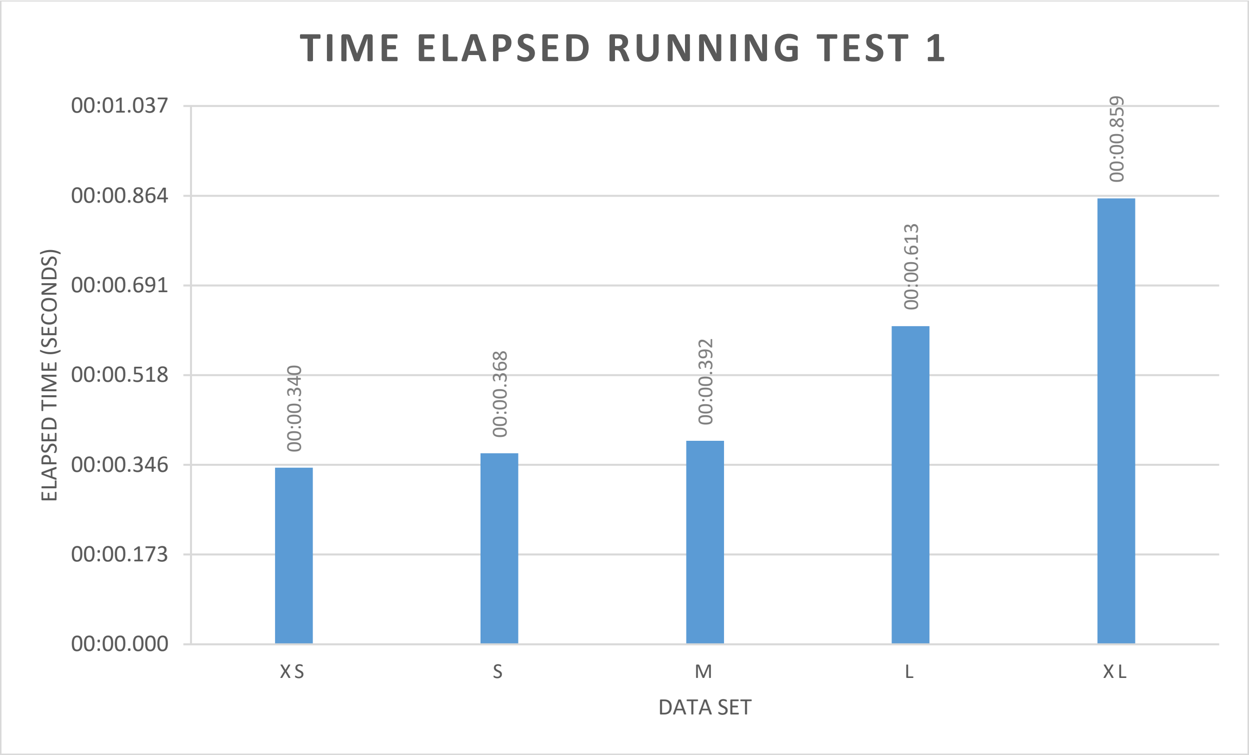


Figure 8. Time spent running Test 1

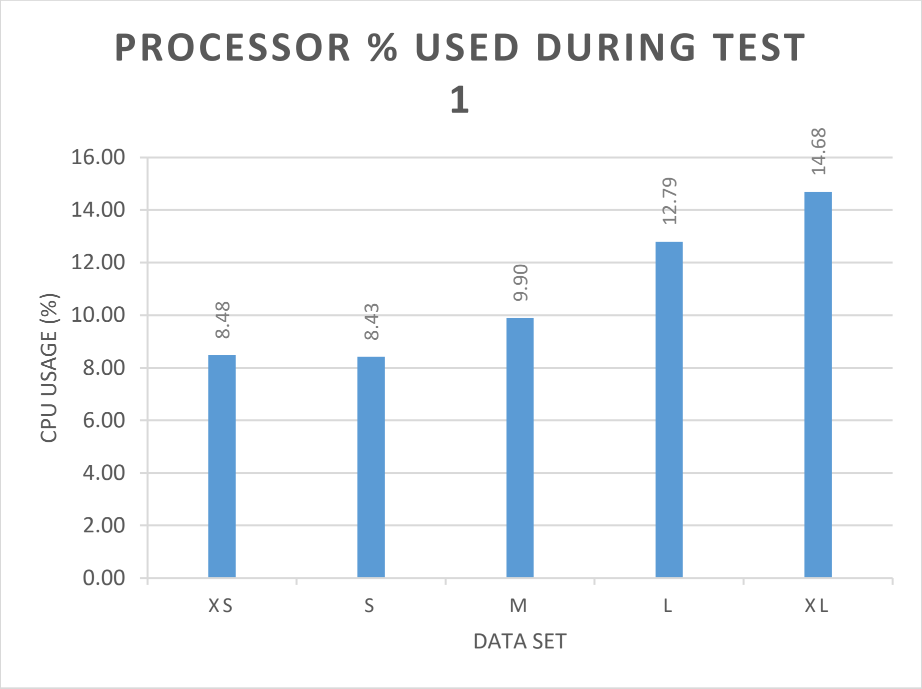


Figure 9. Processor % used during Test 1

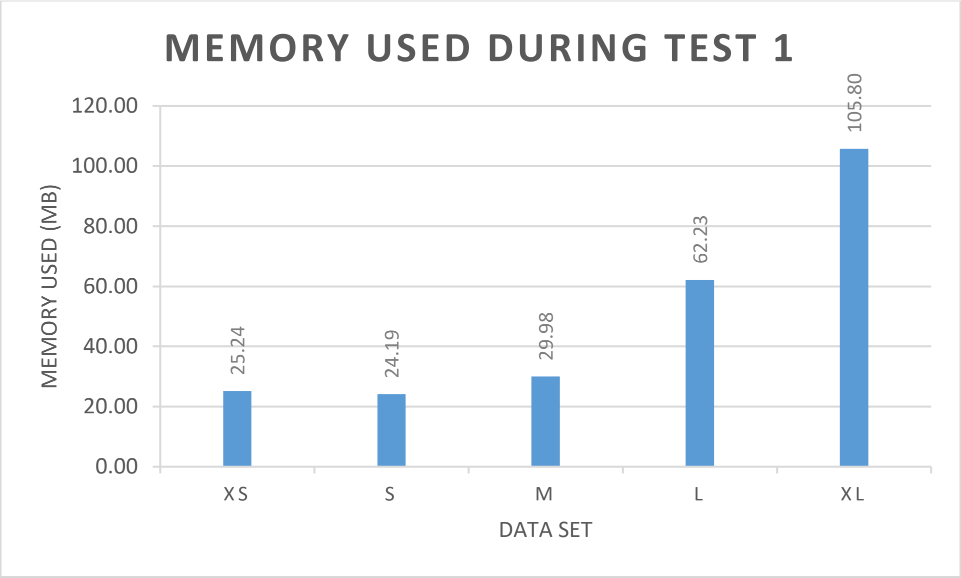


Figure 10. Memory used running Test 1

4.4.2 Time Results

The results from Figure 11 show the amount of time taken for the search to run for all six test scenarios. The results show that the amount of time it will take to perform a search depends on the type of search. If the search is a top-down search, the search time grows exponentially with the size of the data set. In contrast, if a bottom-up search is performed, the search time grows in a near-constant or weak-linear fashion. Since the exponential growth only occurs with a search that experience a top-down query (Tests 2, 4, and 6), these results indicate that MongoDB experiences a slow down when having to perform a graphlookup against a larger number of attached objects. While a bottom-up search will attach two related objects at most per content found, a top-down search could attach up to 110 objects per case found.

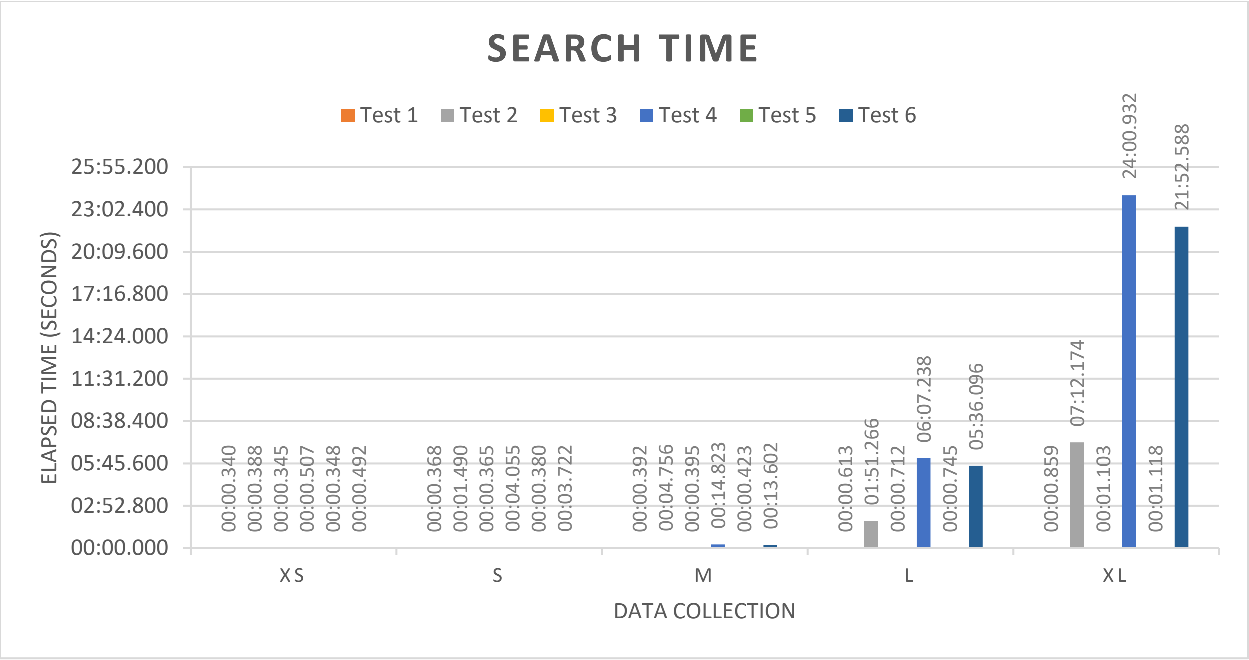


Figure 11. Search time for all Tests

4.4.3 Memory Use and CPU Results

The results in Figure 12 and Figure 13 show how much memory and CPU usage was used during the search. Unlike with the search time results, the amount of resources consumed show a near-constant to weak-linear growth with the growth of the data sets, regardless of direction. With all of the data sets for these test instances experiencing the same growth pattern, this indicates that the hardware used did not contribute to the exponential growth seen in the top-down searches.

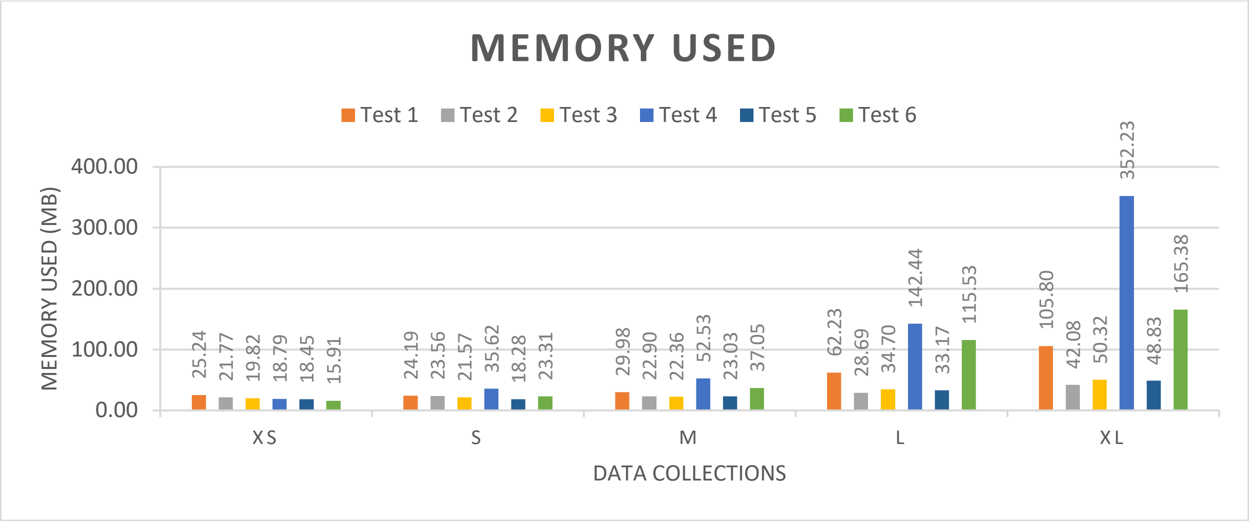


Figure 12. Memory used during Tests

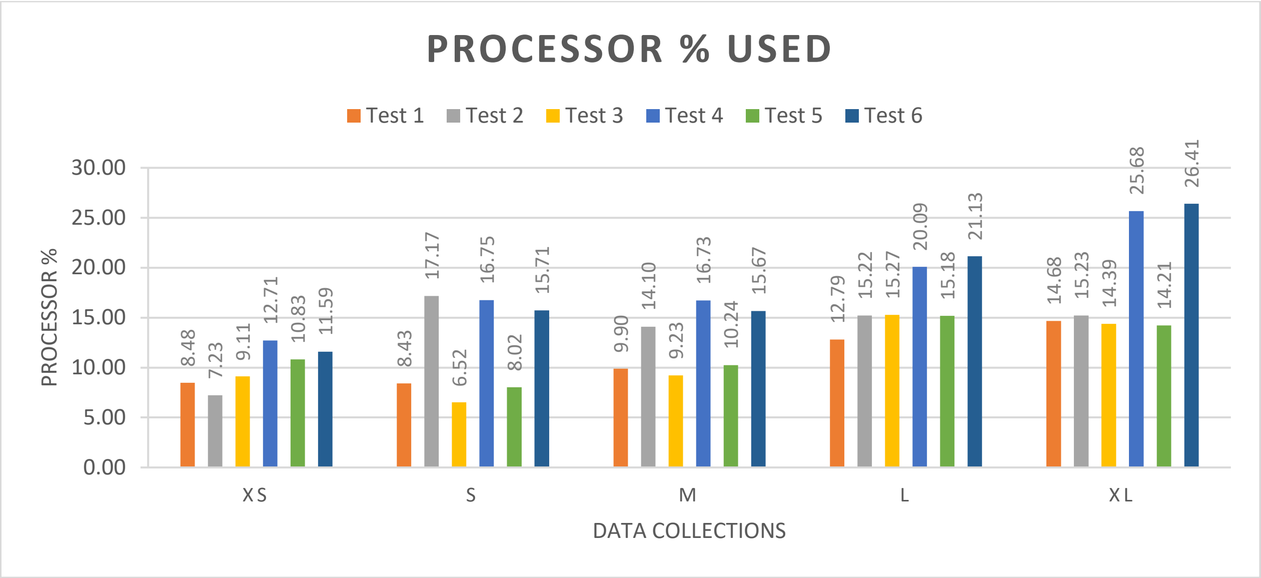


Figure 13. Processor % used during Tests

Chapter 5. Conclusion

This thesis proposed and developed the Electronic Evidence Locker (EEL), an ontology-based NoSQL storage, search, and sharing system for electronic evidence in a cyber and physical crime. First, the proposed ontology provides a formal specification and representation of the categories, properties, concepts, and relations for this domain. Then, the EEL was developed using a NoSQL database for storage and C# .NET Core for the website.

Baseline performance results were produced to measure the growth in the amount of required machine resources caused by increasing the complexity of the search (query) criteria and with increasing the size of the data. Finally, the provided performance evaluations help to show the relationship between the increasing complexity of the search query with various sizes of the data set (e.g., medium, large, extra-large) in terms of computer resources such as memory, CPU utilization, and the required time to perform an exhaustive search.

5.1 Discussion of Electronic Evidence Locker Implementation

While the ontology has been implemented into the project and tested, an interesting observation showed that what made queries either use more resources or run longer ended up not being how many conditional statements were added to a query, but in what direction the search would flow and how many levels the search would have to traverse. One difficulty in evaluating resource usage with the created search method is that how MongoDB performs optimizations for query searches extends beyond the scope of this project. If MongoDB changes how the optimizations are performed in future versions, future results may differ from the presented results.

5.2 Future Work

With the baseline performance recorded, future research can look at improving EEL's performance in several areas. A promising direction is to investigate the trade-off between the benefits and the overhead (memory consumption) of using indices on unstructured data. Another direction would be building the search tool in a dynamically typed language, i.e. Python or Ruby. This area would allow for the creation of new fields dynamically without having to update the web application for every new field added.

References

[1] A. Katal, M. Wazid and R. Goudar, "Big data: issues, challenges, tools and good practices," in *2013 Sixth International Conference on Contemporary Computing (IC3)*, 2013.

[2] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013.

[3] "Key-Value Databases," MongoDB, [Online]. Available: https://www.mongodb.com/databases/key-value-database. [Accessed 21 October 2021].

[4] C. M. Wu, Y. F. Huang and J. Lee, "Comparisons between mongodb and ms-sql databases on the twc website," *American Journal of Software Engineering and Applications,* vol. IV, no. 2, pp. 35-41, 2015.

[5] S. S. Nyati, S. Pawar and R. Ingle, "Performance evaluation of unstructured NoSQL data over distributed framework," in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2013.

[6] L. Obrst, "Ontological Architectures," *Theory and Applications of Ontology: Computer Applications,* pp. 27-66, 2010.

[7] "MongoDB Limits and Thresholds," [Online]. Available: https://docs.mongodb.com/manual/reference/limits/. [Accessed 13 September 2021].

[8] "JSON and BSON," MongoDB, [Online]. Available: https://www.mongodb.com/json-and-bson. [Accessed 21 October 2021].

[9] "Operational Factors and Data Models," [Online]. Available: https://docs.mongodb.com/manual/core/data-model-operations/#collection-contains-large-number-of-small-documents. [Accessed 13 September 2021].

VITA

DANIEL SMITH

Education: M.S. Computer & Information Sciences, East Tennessee State University, Johnson City, Tennessee, 2021

B.S. Computing Major with Information Systems concentration, East Tennessee State University, Johnson City, Tennessee, 2009

Professional Experience: Senior Software Engineer, East Tennessee State University, Johnson City, Tennessee, 2020-Present

Programmer/Analyst II, East Tennessee State University, Johnson City, Tennessee, 2015-2020

Application Support Team, Tele-Optics, Kingsport, Tennessee, 2013-2015

Web Developer, Send the Light Distribution, LLC., Elizabethton, Tennessee, 2012-2013

IT Manager, Restaurant Management Group, Kingsport, Tennessee, 2009-2012