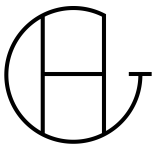

HoGent

Faculteit Bedrijf en Organisatie

Analysis and Design of an Efficient NoSQL Data Storage Schema for an Interlinked Recent Activity Feed

Florian Dejonckheere

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Chantal Teerlinck

Instelling: Open Webslides

Academiejaar: 2017-2018

Tweede examenperiode

Faculty of Business and Information Management

Analysis and Design of an Efficient NoSQL Data Storage Schema for an Interlinked Recent Activity Feed

Florian Dejonckheere

Thesis submitted in partial fulfilment of the requirements for the degree of
professional bachelor of applied computer science

Promotor:
Chantal Teerlinck

Institution: Open Webslides

Academic year: 2017-2018

Second examination period

Preface

Samenvatting

Abstract

Contents

Glossary

databasemanagementsystem Software that manages storage, querying and manipulation of data in a database. 25

List of abbreviations

RDBMS Relational Database Management System. 25

List of Figures

List of Tables

Introduction

0.1 Context

In this age of computers and smartphones, older and deprecated methods of teaching are quickly being replaced by their digital equivalents. Course content has shifted from being printed in full-text on paper, to static slides on an overhead projector, to the digital canvas present in every modern classroom. However, there is a lot more potential to gain from the modernization of educational content. Education is still too often a one-way street, where students are obligated to process the course content without being provided much challenge or activity. This does not allow for any dialogue to take place between students and teachers concerning feedback and improvement of the course material itself.

Technological constraints in current iterations of educational software do not allow this co-creation discourse easily. Material being locked to specific versions of proprietary software is just one of the many problems teachers might encounter when trying to apply this concept in real life.

By utilizing interactive tools and applications, the teacher can engage the students more directly.

By building on modern, open standards, the Open Webslides project (Open Webslides, 2017a) aims to provide a platform that solves these problems. It creates a user-friendly environment where teachers can create courses based on open source technologies and standards, and it allows teachers and students to apply the co-creation narrative easily. This also enables users to share their material not just with their immediate environment but with a much broader educational audience.

The concrete implementation of the Open Webslides platform is built to cater to higher education institutions. This entails that the traffic on the platform will be relatively high yet predictable, and seasonally bound. One of the consequences of this idiosyncratically shaped traffic is the requirement for the platform and infrastructure to efficiently handle these workload peaks.

0.2 Problem statement

The Open Webslides platform incorporates several ways to stimulate spontaneous co-creation between teachers and students. One of the most prominent elements is the *Recent Activity* feed. This reverse chronologically ordered list enumerates the most recent user interactions with the platform and with other users. Feed items range from simple actions such as a user having created or modified course content, to more complex social interactions like a discussion facilitated by comments or a student's changes being incorporated in a teacher's courses.

The size of the activity feed data set is directly correlated to the size and activity of the users. It has the potential to grow explosively, especially in timespans critical to teachers and students such as examination periods. In order for the infrastructure to be able to handle the deluge of the queried information, designing a system that allows for efficient querying and easy scalability is of paramount importance. Further decoupling of this subsystem from the business-critical processes is also important to ensure that downtime of this subsystem has no impact on round the clock availability of course content to the users.

This research thesis will provide a comparative and analytical study of existing NoSQL database solutions for the Open Webslides project to implement an efficient, scalable data storage system in the context of the *Recent Activity* feed. A basic solution is already implemented within the platform, however the proposed data model allows for more flexibility and accommodates any future functionality expansion.

0.3 Research questions

The research in this thesis is focused on finding an answer to three main research questions:

1. What frameworks and software packages currently exist in the industry to store structured non-relational graph or document data?
2. How is the social graph as introduced by the Open Webslides' Recent Activity feed conceptually and logically structured and how is this data consumed?
3. What NoSQL data store is the most appropriate and efficient data store to store this social graph?

Determining and exploring the NoSQL database landscape will provide us with a general idea of the current state of affairs. This knowledge will then be utilized to answer the

second research question in form of concrete data models and queries. Finally, The first two questions will then lead into an analysis to answer the last research question. It will also provide a practical approach for the Open Webslides development team to take into account this research paper when developing the platform in the future.

0.4 Research goal and objectives

Since this research contains both a general, more theoretical and a practical, use-case oriented panel, the goal of this thesis is dual. First, a comparative study of the NoSQL horizon, its applications and vendors, which might be of interest for a broader audience. Second, a concrete recommendation of a NoSQL data store for the Open Webslides project.

0.5 Expected results and conclusions

Expected results and conclusions

In chapter 1 an overview will be presented of current research, applied solutions and other related work in the research domain based on a literature study. This literary review will then be used to give a broad and global overview of the research domain, introducing various relational and NoSQL concepts and explaining the motivation behind NoSQL and big data in chapter 2.

Chapter 3 will clarify the analytical research approaches used in this thesis to attempt to formulate an answer to the research questions. The research is split into three distinct parts. The first part, Chapter 4 will evaluate existing NoSQL data stores, and select the data stores used for comparison in this thesis. The second part in chapter 5 proposes the conceptual and logical data model that may be used in the Open Webslides project, and presents examples of queries that may be frequently executed on the data set. Chapter 6 combines the proposed data models and reference queries in order to perform and discuss performance benchmarks on the data stores selected in chapter 4.

Finally, chapter 8 will present a general conclusion and summarize the research, answering the research questions.

1. State of the Art

Ever since the rise of the NoSQL databases in 2009 (Sadalage & Fowler, 2012) it has been a subject for vigorous academic and professional research. The contrast with relational databases, optimal use cases, performance and scalability are only some of the aspects that have been analyzed with great regularity. This chapter will summarize previous publications relevant to this thesis.

The book by Sadalage and Fowler (2012) provides an excellent entry into the world of NoSQL. It explains the motivation behind the use of NoSQL techniques, and how this differs from relational data storage. Furthermore it introduces the segregation of NoSQL data stores into four main categories: key-value, document, column and graph databases. Second, Sadalage and Fowler touch the concept of polyglot persistence. This describes the concept of an application using multiple types of data stores to store heterogeneously structured data. This technique is relevant in particular to this thesis, as the describe data schema only relates to one of the database management systems integrated in the Open Webslides platform. The second part of the book provides a more practical approach to using polyglot persistence in an enterprise application. The authors have written down many pointers and guides in order to pick the right database for a particular use case.

There have already been numerous studies to differentiate the different types of NoSQL databases and comparative studies between NoSQL database systems. Nayak, Poriya, and Poojary (2013a) present a fifth NoSQL category: object-oriented databases. Other studies such as Moniruzzaman and Hossain (2013) and Maroo (2013) provide a feature-based comparison for various NoSQL vendors and database systems. Finally, the similarity and resemblance of relational and NoSQL data stores is also a well-researched topic in current literature. Studies and surveys such as Mohamed and Ismail (2014) and Cattell (2010) tackle this subject in great detail.

Grolinger, Higashino, Tiwari, and Capretz (2013) present a use-case based approach to comparing different NoSQL and NewSQL data stores. The survey incorporates a feature-based comparison over different aspects such as querying, scalability and security, and analyzes these concepts in the context of a select number of NoSQL data stores.

Hecht and Jablonski (2011) provides a feature-based comparison of different NoSQL database types and vendors. The researchers compare the data model, querying access, concurrency, partioning and replication. The paper uses a duality-based approach, where a minus indicates that the feature is not supported by the database system, and a plus if the feature is supported. The paper also presents the problem of a lack of unified querying interface for NoSQL databases. Furthermore, the importance of choosing the right NoSQL database type for the use case is emphasized, however Hecht and Jablonski do not present a specific case study.

The proceedings of the 2013 IEEE International Conference on Big data by Kaur and Rani (2013) describe the theoretical modeling and querying of SQL and NoSQL data stores. The paper then proceeds with a case study of a social networking site similar to Slashdot (Malda & Bates, 1997). Starting from an entity-relationship diagram (ERD), the researchers then proceed by modeling the entities in both a document and a graph database. Finally, a set of seven queries related to the use case is then drawn up and compared for the PostgreSQL, MongoDB and Neo4j data stores.

Zhao (2015) explores the use of NoSQL data stores to store huge amounts of observational data generated by astronomical research. It briefly discusses using filesystems and relational data stores, before comparing NoSQL alternatives. A concrete data model to store the astronomical data in a MongoDB data store is then presented, together with eight scenarios and queries that may be used in a production system. Furthermore, performance measurements of MongoDB are also analyzed. Data insertion, querying and deletion using the aforementioned data scheme and real observational data are used in this section.

The proceedings of the AGILE 2015 conference by Schmid, Galicz, and Reinhardt (2015) present an overview of selected SQL and NoSQL databases, focusing on the geo-functionalities of the systems. It uses performance tests between two document-based NoSQL data stores (MongoDB and CouchBase). The researchers conclude that geospatial calculations in NoSQL database systems are still only supported for basic queries. Relational databases still perform superior to NoSQL databases in small to larger data sets for queries with geo-functions. However the NoSQL response time only increases slightly relative to data set size.

The technical report by Barahmand, Ghandeharizadeh, and Li (2015) quantifies the scalability of MongoDB and HBase for processing simple operations using the social networking benchmark BG (Barahmand & Ghandeharizadeh, 2013). It considers both horizontal and vertical scalability of the data stores using the Social Action Rating (SoAR) introduced by the benchmarking tool. In order to perform these benchmarks, two logical data models for the database design are presented. The report concludes that while both data stores scale superlinearly, their speedup is limited by the resources of a few nodes out of many becoming fully utilized.

Another performance-based study written by Abramova, Bernardino, and Furtado (2014) compares five popular NoSQL databases (Cassandra, HBase, MongoDB, OrientDB and Redis) using the Yahoo! Cloud Serving Benchmark (Cooper, Silberstein, Tam, Ramakrishnan, & Sears, 2010). The study compares read and write query performance. It concludes that over the five compared data stores, MongoDB, Redis and OrientDB are more read-optimized, and Cassandra and HBase are more update-optimized.

A more query-oriented study was performed by Zhou, He, Sheng, and Wang (2013). The paper considers both the academic and the industry definition and description of data models and system architectures. The researchers identify two kinds of searching approaches: the MapReduce-oriented and the SQL-like querying.

The work of Atzeni, Bugiotti, Cabibbo, and Torlone (2016) dives deeper into the world of non-relational data modeling. The paper investigates how traditional data modeling can be used in the context of schemaless and heterogeneous data stores. Atzeni et al. propose NoAM (NoSQL Abstract Modeling), an abstract data model to describe NoSQL databases based on the common surfaces of the various data store types. This technique can be used to describe system-independent application data and later to implement this in the specific data stores, taking advantage of the various target system idiosyncrasies. Further articles by the same authors (Bugiotti, Cabibbo, Atzeni, & Torlone, 2014) expand upon this abstract data model to present a database design methodology for NoSQL systems.

The main differences between this study and the previous studies are:

1. Many studies have been conducted to understand the motivation between the NoSQL principles and the shift from relational data stores. The division of NoSQL data store types into four most commonly recognized categories and elaboration upon this is usually also a topic in these studies. This research paper builds upon that knowledge, providing only a brief introduction in the world of NoSQL and NewSQL concepts.
2. Some of the previously mentioned research papers also discuss a case study applied to a specific use case. This is mostly related to business critical systems that store and process large volumes of data. The use case described in this thesis is very specific in that it's a complementary subsystem that does not affect critical data. Consequently, certain comparative attributes such as security and availability are not considered in this research.

This thesis aims to provide a case study of data storage the Open Webslides (2017a) platform. Several use case based surveys and studies already exist, however they aim at replacing a relational database in an application with a NoSQL database without bringing polyglot persistence into account. Sadalage and Fowler (2012) is one notable exception in this aspect. In the case of Open Webslides, the NoSQL data store only complements the relational database and does not fulfill a critical function. Therefore, several constraints such as security and availability differ in interpretation from existing studies.

2. Overview

2.1 Relational data stores

Relational Database Management System (RDBMS) have been the de facto standard for over two decades to efficiently store and query information in a wide variety of native and web applications. These data stores are based on the relational model devised by Edgar Codd. The data in the system is presented as relations: a collection of data consisting of rows and columns. The user of the database can query and manipulate the data using relational operators.

Codd presented thirteen rules for a database in order for it to be considered a *relational database*. However, many of the modern database systems do not adhere strictly to all of these rules. More commonly, a relational database is defined as a database that exposes its information using a collection of rows and columns.

Relational database management systems provide certain guarantees during database transactions. Reuter and Härder introduced the concept of ACID: an acronym referring to a set of transactional properties.

1. **Atomicity**
   Each transaction is "all or nothing", either the transaction completes successfully and the data is mutated in an atomic way, or the transaction fails entirely and none of the data in the database is modified.
2. **Consistency**
   Each transaction, when successful, only commits legal results. This entails that the database is always in a consistent state.
3. **Isolation**

Actions within a single transaction are not visible to other, concurrently running transactions.

4. **Durability**
   Once a transaction has completed successfully and been committed to the database, the system must guarantee that these results survive any subsequent malfunction.

2.2 NoSQL data stores

In the early 2000s, a totally different concept of data storage was popularized. NoSQL data stores provide a system of storage that is "non SQL" or "non relational". More recently, the NoSQL denomination has also been explained as "not only SQL", pointing on the fact that some data stores provide a different interface besides SQL. Many databases expose a REST API as primary execution interface. Other data stores have devised their own binary communication protocol, such as the Bolt protocol (Neo Technology, 2007a) for the Neo4j graph data store.

The use of non-relational data stores was motivated by the needs of Web 2.0 companies such as Facebook and Google. NoSQL provides a way to store massive amounts of data in a flexible way. The architecture of such systems is usually more simple, and focuses more on horizontal as opposed to vertical scalability. Data is not stored in rows and columns as is the case in the relational model, but rather in a different data structure. Nayak et al. (2013a) divides the data models used by NoSQL data stores into five categories.

2.2.1 Key-value

Key-value data stores are simple in design, yet powerful and efficient when used correctly. A key-value data store allows the user to store schemaless data using a plain, unique key, creating a *key* and *value* pair. Similar to hash tables, the values are stored and queried using the provided key which is used as an index. Modern key-value data stores prefer high scalability over consistency, entailing that more advanced ad-hoc querying and analytical operations on the data such as joins are not available.

2.2.2 Column-oriented

Column-oriented data stores are designed to store data by column rather than by row. This kind of NoSQL data store is more similar to relational database systems than the other categories. In column-oriented data stores, each key is associated with a set of attributes, stored in columns. Concretely this means that the data is indexed by column value, rather than by row.

2.2.3 Document

Document databases store their data in documents, indexed by a unique key. This is technically a subclass of key-value stores, however the difference lies in the interpretation of the data. Since documents are schemaless, each document may have a comparable structure, or a completely different one. In contrast with key-value data stores, the value (document) is not opaque to the database management system, but is interpreted and used for query optimization.

2.2.4 Graph

As the name suggests, graph data stores keep their data in the form of a graph. The graph is made up of nodes and edges, the former being the entities containing the data and the latter being the relations between these entities. Graph data stores use a technique called *index-free adjacency*, where every node contains a logical pointer to the adjacent node. This makes graph traversal a very fast operation. Graph databases are ACID compliant.

2.2.5 Object-oriented

Object-oriented data stores represent the data as an object, closely resembling the concept of an object in object-oriented programming languages. This puts object-oriented data stores much closer to the programming environment than other database systems. It provides all features inherent to object-oriented languages, such as data encapsulation, inheritance and polymorphism. The concepts of class, class attributes and an object can be mapped onto the relational concepts of a table, columns and a tuple. This concept of data storage makes software development more flexible.

2.2.6 Multi-model

Data stores are generally built and optimized around one data model. However, there exist multi-model data stores as well. These database systems allow storing data using any of the different data models mentioned before, while integrating these into the same server package. One example of such a data store is OrientDB (OrientDB Ltd, 2010). Multi-model data stores support and stimulate the principle of polyglot persistence, where an application utilizes more than one type of data store, while reducing its operation complexity.

Most NoSQL data stores are built around the concept of *eventual consistency*. This is a consistency model that dictates that all accesses to a particular piece of data will eventually return the last updated value. This principle is broadly implemented in distributed computing systems. Systems providing this property are also classified as BASE: Basically Available, Soft state, Eventual consistency. In contrast to the ACID properties, systems built around the BASE principles prefer availability over consistency.

In 2000, Eric Brewer presented a conjecture known as the CAP theorem (Brewer, 2000). This conjecture, later formally proven (Gilbert & Lynch, 2002a, 2), asserts that it is impossible for a distributed data store to exhibit more than two out of three of the following properties:

1. **Consistency**: Every read operation receives the most recent write result
2. **Availability**: Every request receives a non-error result
3. **Partition tolerance**: The system continues to work despite failure to communicate between nodes

The CAP theorem states that when a network partition is present, the database developer has to choose between providing consistency or availability. Note that *consistency* as defined by the CAP theorem is not the same concept as consistency as described by the ACID properties. Database systems respecting the ACID guarantees choose consistency over availability, while database systems built on the BASE principle generally choose availability over consistency.

NewSQL is a type of relational database management system providing scalability similar to NoSQL data stores, while still maintaining the ACID guarantees. Since this a blanket term, many different architectures exist in NewSQL databases. However, one common feature is their relational model inheritance and their ability to utilize SQL as query interface.

3. Methodology

Analyzing and comparing every NoSQL data storage solution is not feasible due to the sheer number of competing products. Therefore we have made a selection of the most popular NoSQL data stores in the different NoSQL categories that are included in the research. This selection is mainly motivated by a list of the most popular databases kept up to date by Solid IT (2018). For every category, we select the most popular databases by reported score, which is in turn based on several other criteria. DB-Engine Ranking attempts to estimate the overall popularity of the data store on the Web.

Next, the selected data stores are committed to a general comparison. In this comparative study we analyze every solution based on various aspects. The data model, which is the main driving force behind many other aspects, is one of the main focus points. We also focus on the querying capabilities, scaling, partitioning and replication, consistency and concurrency control. Certain aspects are not discussed in detail due to their irrelevance to the presented use case. Examples of these criteria include security, authentication and auditing.

In the following chapter, the conceptual and logical data model of the Open Webslides project is presented. Building upon this, a physical data model is discussed and implemented for the various categories of NoSQL data stores that were selected in the first part of the research. Finally, the characteristic data access flow within the application is examined, and a few reference queries are introduced. These queries are examples of queries that could typically be used to retrieve data from the data store as part of the normal operating procedures of the Open Webslides application.

Finally, the implementation of the data model and the reference queries are used in the last part of the research as a base for qualitative benchmarks. Since the Open Webslides

application is built on Ruby and Ruby on Rails, the benchmarks will be implemented in a Ruby on Rails application, making use of the available Ruby language bindings to the various data stores examined. For every data store used in the benchmark, the most popular ORM gem is used as determined by the RubyGems repository Related work and previous NoSQL database benchmarks are also considered in this part, however it is referenced only as a baseline due to the fact that previous work does not cover the specific use case and NoSQL data stores studied in this thesis.

4. Data stores

4.1 Selected data stores

Due to the large number of NoSQL data stores it is not possible to consider all of them for this research. In every applicable NoSQL category the most popular data stores are selected, where popularity is based on certain predetermined parameters. Solid IT (2018) maintains a list of database systems ranked by popularity, based on parameters such as number of mentions on websites, Google Trends and availability of relevant jobs. The list includes not only NoSQL databases but also other types of data storage systems such as relational database systems. This ranking, named DB-Engine Ranking, is used primarily to determine which data stores are the most interesting to consider in this research. Due to licensing concerns regarding the Open Webslides project, database systems that do not fall under a free license as classified by the Free Software Foundation (1985) will not be considered.

Certain categories of NoSQL data stores will not be considered. Key-value stores will not be accounted for due to the relative simplicity of this type of data store. The main use case of key/value stores is not storing more complex information, rather the emphasis lies more on scalability and consistency. Processing complex queries that consist of the NoSQL equivalent of relational joins is not efficient in key-value data stores. Implementing the proposed data model and queries in such a system is a more ambitious task that does not lie within the scope of the research.

Furthermore, NewSQL data stores will not be included in the research either. As described in chapter 2, NewSQL aims to provide scalable performance similar to that of NoSQL while still guaranteeing ACID properties. Since ACID is not a major concern for this use case, NewSQL does not provide significant advantage over NoSQL, and will therefore be

omitted from the comparison.

Similarly, object-oriented and multi-model database such as OrientDB (OrientDB Ltd, 2010) are not within scope of this research. The comparison will thus aim at document, column and graph data stores.

The most popular document database systems according to the DB-Engine Ranking are MongoDB (MongoDB Inc., 2009), CouchDB (Apache Software Foundation, 2005a) and Couchbase (Couchbase, Inc., 2010). While Couchbase is technically a multi-model data store, it is being ranked as a document database. However, Couchbase is a conceptual merge between CouchDB and Membase, and mainly adds improved scalability, clustering and auto-sharding. Couchbase will not be investigated upon in a practical capacity, however it is considered as an option to increase multi-host scalability. This leads to MongoDB and CouchDB being the contestants in the document database system category.

As (wide) column data stores, Cassandra (Apache Software Foundation, 2008) and HBase (Apache Software Foundation, 2005c) are among the highest ranked databases. These two database systems will be selected as candidates for the second NoSQL category.

Finally, as sole graph database the Neo4j system (Neo Technology, 2007b) stands out greatly over competitors in the ranking. This database system will be analyzed as only graph database in this research.

In conclusion, the data stores included in the study will be MongoDB and CouchDB as document databases, Cassandra and HBase as column databases and Neo4j as graph databases.

4.2 Comparative study

Data stores and databases can be compared and analyzed using many quantitative and qualitative aspects. In this thesis we propose a selection of criteria based on the usefulness applied to the studied use case. Since the landscape and feature-set of NoSQL data stores is changing on a weekly base, the impact of these technologies must be carefully considered in order to reach a durable conclusion in this research.

The features of a data store that are taken into account in this comparative study are:

- Persistence
- Querying capabilities
  - Language
  - Protocol
  - MapReduce
  - SQL
- Language bindings
- Integrity model
- Atomicity

- Consistency
- Isolation
- Durability
- Transactions
- Partitioning
- High Availability
- Concurrency
- Replication
- License
- Commercial support

Certain aspects are not relevant to the presented use case because of various reasons. Aspects omitted from the study are:

- Security
- Compression
- Full-text search
- Geospatial functionality
- Cloud hosting

5. Data model

5.1 Logical data model

In this section, a data model is explained in detail and represented in an Entity Relationship (ER) diagram. The data model closely follows the already existing data model of the Open Webslides project, however certain irrelevant attributes are omitted from the schema to improve readability and simplify abstraction.

The entity that will most likely be the entrypoint for the reference queries is `User`. This entity contains information pertaining to the user, such as email address, first name and last name.

On the platform, a user can create and modify course content in an online editor. This course content is abstracted in the database as the `Topic` entity. The `Topic` entity does not contain all data regarding the course content, rather it holds the metadata and a logical pointer to the actual course content data, which is stored on the filesystem in a git repository. Per illustration, the `title` and `description` attributes are included in the described data model, because these two attributes are the only metadata relevant to the *Recent Activity* feed.

The user has three distinct possibilities for (co-)creation on course content. This is only from a technical perspective, we will not go further into the conceptual implications of co-creation. First, the user can directly modify the course content on the `Topic` entity. To simplify the data model, only authors can modify `Topic`, which covers the one-to-many relationship between `User` and `Topic`. The second option is creating annotations on the topic. This allows the user to attach notes to specific content on the document. The `Annotation` entity contains a logical pointer to the annotated content, which is

omitted from the schema. The final possibility to integrate user content into a topic is by adding comments. In contrast to annotations, comments can also point to other comments, which allows interaction and conversation between multiple users, and effectively enables dialogue between students and teachers.

5.1.1 Recent Activity feed

In the previous section a basic data model was described. This model does not include the entity or entities necessary to enable the *Recent Activity* feed functionality. Since the research questions of this thesis revolve around storing this information in a NoSQL database, a clear separation of relational data and non-relational data is desirable.

Every item in the *Recent Activity* feed is represented by one entity, `FeedItem`. This entity has three attributes, as inspired by the Resource Description Framework (RDF) triple: `subject`, `predicate` and `object`. The subject refers to the user which is executing the action. The object points to the entity the user is manipulating, be it a topic, comment or annotation. These first two attributes are actually pointers to other entities within the database. The predicate describes the action being executed, and is a value of a predetermined enumeration.

Predicates can range from simple values such as `CREATED` or `UPDATED` to more conceptually complex values such as `REACTED_TO_ANNOTATION`. The predicate enumeration is chosen so that future extension with additional predicate types is simple. Concretely this means that the value is stored as a `string`, while higher level abstractions like ORMs and database mappers will enforce data type consistency.

Some examples of these triples are:

- `User X created Topic Y`
- `User X updated Topic Y`
- `User X commented on Topic Y`
- `User X responded to User Y's comment on Topic Z`
- `User X created an annotation Topic Y`

5.2 Physical data model

5.2.1 Document-oriented data model

A document-oriented data store is used to store and query semi-structured data. Data is stored as documents, an unstructured and extensible data format. CouchDB stores its documents as JSON, whereas MongoDB utilizes Binary JSON (BSON) as primary data format. The advantage of storing data in JSON or BSON is that objects and structures of many programming languages can be deduced directly into this format, rather than using a mapping or translation layer.

5.2.2 Column-oriented data model

Column-oriented databases store their data using columns.

5.2.3 Graph-oriented data model

Graph-oriented databases such as Neo4j have their data stored in a network structure. Logical entities are mapped to graph nodes, and relations to edges between the graph nodes. An edge can be uni- or bidirectional: the nodes will indicate this as incoming, outgoing or both. Graph nodes also contain properties, akin to the entity attributes on the ER diagram. On the first sight, this structure lends itself excellently for modeling social interactions. Every graph node can have many incoming and outgoing edges, and the structure of the network is not predetermined. Due to trees being a subset of graphs, graph data stores are also very good in representing hierarchical data. Therefore, if the data is structured similar to a tree or a graph, it can best be stored in a graph database rather than tabular or document data in order to reduce the object-relational impedance mismatch.

An additional advantage of graph databases is that the relationship between two entities can also contain properties. This enables the developer to easily add metadata to any object in the database, whether it is a vertex or an edge.

5.3 Reference queries

6. Comparative study

7. Opportunities

8. Conclusion

A. Research proposal

The subject of this thesis is based upon a research proposal that was graded by the promotor in advance. This proposal has been included as an attachment.

A.1 Introduction

The Open Webslides (2017b) project provides a user-friendly platform to collaborate on webslides - slides made with modern web technologies such as HTML, CSS and JavaScript. One of the core features this application provides is *co-creation*. The co-creation aspect manifests itself in several forms within the application; annotations on slides and a change suggesting system resembling GitHub's pull request feature are the main mechanisms. Because of the inherent social nature of co-creation, a basic notifications feed was also implemented. This feed is tailored to the user, and reflects the most recent changes, additions and comments relevant to the slide decks the user is interested in.

However, at the moment the functionality implemented in the system contains only the bare necessities. The module will be expanded in the future, and doing so requires a structural and conceptual rethinking of how the notifications are generated, stored and queried. The size of the dataset is also expected to grow linearly with user activity, therefore scalability is a requirement as well.

This paper has two research questions.

1. What frameworks and software packages currently exist in the industry to store structured non-relational graph or document data?
2. How is the social graph provided by the Open Webslides' notification feed structured

and how is this data consumed?

Answering these questions is paramount for the final section of the paper, which describes a data storage mechanism that performs well given the functional requirements of the project's data flow.

A.2 Use case

This thesis is a study of NoSQL data storage techniques applied to the Open Webslides (2017b) project. The online, interactive platform that the project provides includes a list of notifications in reverse chronological order, tailored to the user. This feature is called the *social feed*. It enumerates the most recent social activity on the platform. For example, if a user updates a slide deck, the user's friends would be able to find a change notification in their respective social feeds.

From the perspective of the application code that generates the social feed, the user, slide deck and notification should be considered separate entities. The notification itself has relations to the other entities: the author (the *subject*), the slide deck (the *object*), along with the *predicate* property that provides information on what operation was performed (for example creating or updating a slide deck).

This data model is also characterized by the *write-once read-many* nature of the information; once the notification has been generated, it does not need to be modified again. It will also only be queried in a very specific way: the application server will always attempt to retrieve the most recent notifications starting from the user entity. This principle is an important aspect to take into consideration in the choice of data storage mechanism.

A.3 State-of-the-art

In current literature, studies such as Moniruzzaman and Hossain (2013), Nayak, Poriya, and Poojary (2013b) and Dayne Hammes and Mitchell (2014) have already analyzed the disparity between traditional relational database systems and NoSQL stores. However, the conceptual and technical difference between these data storage models will not be scrutinized any further, since this paper presents a data storage solution applied to the social notification feed of the Open Webslides (2017b) project.

There are already many existing free and commercial products for the storage of NoSQL data, such as Redis (Sanfilippo, 2009), CouchDB (Apache Software Foundation, 2005b), MongoDB (Inc., 2009) and Neo4j (Technology, 2007). Finding the right database model for this use case (section A.2) is one of the hurdles this paper intends to handle. Zhao (2015) describes the development of a messaging system for astrophysical transient event notifications. Part of this paper is a qualitative comparison between document-based NoSQL storage solutions fit for this particular use case. We expect this paper to provide

a solid base of reasoning in order to find a scalable and efficient solution for resolving similar computational challenges.

The goal of this paper is to provide a performant, scalable and maintainable data storage schema for the Open Webslides (2017b) platform, regarding the linked social graph that powers the *Social Feed* functionality present in the platform.

A.4 Methodology

First, a range of industry-standard NoSQL database management systems such as MongoDB (Inc., 2009), HBase (Apache Software Foundation, 2005d) and Neo4j (Technology, 2007) will be qualitatively analyzed. Three of the five types of NoSQL database types (Nayak et al., 2013b) will be included in the study: column-oriented, document based and graph databases. Criteria for comparison include how the database management system concretely stores its data on disk, the query format and specific programming language bindings. Another important aspect is the distributed nature of many NoSQL databases. Using Brewer's conjecture (Gilbert & Lynch, 2002b, 2) – often called the CAP theorem – the existing types of data storage systems will be examined and summarized. There is also a practical factor present in the research; this includes the license of the project, its active maintainability and future prospects. Common types of NoSQL databases include key-value store, column-oriented, document store and graph databases (Nayak et al., 2013b). This paper will provide a short introduction to these types, before proceeding with the type that fits our use case the most.

Second, the data model specific to the Open Webslides project will be examined. We will start from the data model that is already implemented in the current iteration of the platform. At the time of writing, the existing base implementation of the social notification feed only contains two types of notifications. This paper will try to extrapolate this concept into a more generalized, abstract system in which developers can easily plug additional notification types. The physical properties of the data model will also be taken into account: the data will be written to the data storage only once, but read many times. It is also highly interlinked information, as a notification will always relate to one or more users as a subject, and a target object as well – most likely a slide deck or collection of slide decks. These links need to be maintained, and efficiently reconstructed when queried.

Finally, a sample dataset will be constructed using the aforementioned detailed analysis. Empirical testing will be conducted against multiple database management systems, and the results will be summarized and interpreted. Various information flows will be tested; however, the most important process remains efficiently querying the stored data.

Using the comparative study of storage engines, data model analysis and the empirical results an implementation plan will be constructed. This plan will serve as a recommendation for future development.

A.5 Expected results and conclusions

The NoSQL ecosystem, unlike relational databases, is headed towards specialization, so different solutions are headed in different directions (Maroo, 2013). In this paper, we expect to find one type of NoSQL database that is a better fit for the Open Webslides use case, in clear contrast with the other types of storage engines. Due to the inherently highly interlinked nature of the stored data, we suspect a graph-based database management system to provide most advantages, and generally the most performant experience.

This expectation is amplified by the availability and good community support of Ruby bindings to the most popular graph database management systems.

Since the platform being discussed only caters to a small to medium user base, we do not expect the need to scale horizontally beyond one instance. However, the vertical scalability is still a topic for discussion, and we expect to determine the computational order of magnitude in order to efficiently query the given dataset during this study.

Finally, the implementation plan should describe a concrete roadmap, stretching over a development period with a baseline expectation of one to three months. Roll-out of this mechanism should also be included in this plan.

We also expect that this thesis will provide a good reference to a further stable, scalable and extendable implementation of the *social feed* feature in the Open Webslides (2017b) project as outlined in section A.2.

Bibliography

Abramova, V., Bernardino, J., & Furtado, P. (2014). Which NoSQL Database? A Performance Overview. *Open Journal of Databases*, *1*(2).

Apache Software Foundation. (2005a). CouchDB. Retrieved from https://couchdb.apache.org/

Apache Software Foundation. (2005b). CouchDB. Retrieved from https://couchdb.apache.org/

Apache Software Foundation. (2005c). HBase. Retrieved from https://hbase.apache.org/

Apache Software Foundation. (2005d). Neo4j. Retrieved from https://hbase.apache.org/

Apache Software Foundation. (2008). Cassandra. Retrieved from https://cassandra.apache.org/

Atzeni, P., Bugiotti, F., Cabibbo, L., & Torlone, R. (2016). Data Modeling in the NoSQL World. *Computer Standards & Interfaces*.

Barahmand, S. & Ghandeharizadeh. (2013). *BG: A Benchmark to Evaluate Interactive Social Networking Actions*. University of Southern California.

Barahmand, S., Ghandeharizadeh, S., & Li, J. (2015). *On Scalability of Two NoSQL Data Stores for Processing Interactive Social Networking Actions*. University of Southern California.

Brewer, E. (2000). Towards Robust Distributed Systems.

Bugiotti, F., Cabibbo, L., Atzeni, P., & Torlone, R. (2014). Database Design for NoSQL Systems. In *International Conference on Conceptual Modeling* (pp. 223–231).

Cattell, R. (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Rec. 39*(4), 12–27.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). *Benchmarking Cloud Serving Systems with YCSB*. Yahoo! Research.

Couchbase, Inc. (2010). Couchbase. Retrieved from https://www.couchbase.com/

Dayne Hammes, H. M. & Mitchell, H. (2014). *Comparison of NoSQL and SQL Databases in the Cloud*. Southern Association for Information Systems.

Free Software Foundation. (1985). Free Software Foundation. Retrieved from https://www.fsf.org/

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

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

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data Management in Cloud Environments: NoSQL and NewSQL Data Stores. *Journal of Cloud Computing: Advances, Systems and Applications*, *2*(1), 22. doi:10.1186/2192-113X-2-22

Hecht, R. & Jablonski, S. (2011). NoSQL Evaluation: A Use Case Oriented Survey. In *2011 International Conference on Cloud and Service Computing* (pp. 336–341).

Inc., M. (2009). MongoDB. Retrieved from https://www.mongodb.com/

Kaur, K. & Rani, R. (2013). Modeling and Querying Data in NoSQL Databases. In *2013 IEEE International Conference on Big Data* (pp. 1–7).

Malda, R. & Bates, J. (1997). Slashdot. Retrieved from https://www.slashdot.org/

Maroo, T. (2013). *Handling with Dynamic, Large Data Sets - NoSQL a Buzzword or Savior?* JECRC Foundation.

Mohamed, M. A. & Ismail, M. O. (2014). Relational vs. NoSQL Databases: A Survey. *International Journal of Computer and Information Technology*, *3*(3).

MongoDB Inc. (2009). MongoDB. Retrieved from https://www.mongodb.com/

Moniruzzaman, A. B. M. & Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application*, *6*(4).

Nayak, A., Poriya, A., & Poojary, D. (2013a). Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, *5*(4).

Nayak, A., Poriya, A., & Poojary, D. (2013b, March). Type of NoSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems (IJAIS)*, *5*(4).

Neo Technology. (2007a). Neo4j. Retrieved from https://boltprotocol.org/

Neo Technology. (2007b). Neo4j. Retrieved from https://neo4j.com

Open Webslides. (2017a, March). Open Webslides. Retrieved from http://openwebslides.github.io/

Open Webslides. (2017b). Open Webslides. Retrieved from http://openwebslides.github.io/

OrientDB Ltd. (2010). OrientDB. Retrieved from https://orientdb.com/

Sadalage, P. J. & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Pearson Education.

Sanfilippo, S. (2009). Redis. Retrieved from https://redis.io/

Schmid, S., Galicz, E., & Reinhardt, W. (2015). Performance Investigation of Selected SQL and NoSQL Databases. In *AGILE 2015*.

Solid IT. (2018). DB-Engine Ranking. Retrieved from https://db-engines.com/en/ranking

Technology, N. (2007). Neo4j. Retrieved from https://neo4j.com

Zhao, Y. (2015). *Event Based Transient Notification Architecture and NoSQL Solution for Astronomical Data Management* (Doctoral dissertation, Massey University).

Zhou, L., He, K., Sheng, X., & Wang, B. (2013). A Survey of Data Management System for Cloud Computing: Models and Searching Methods. *Research Journal of Applied Sciences, Engineering and Technology*, *6*(2), 244–248.