Eastern Illinois University

The Keep

Masters Theses

Student Theses & Publications

Summer 2020

A Study on Real-Time Database Technology and Its Applications

Geethmi Nimantha Dissanayake
Eastern Illinois University

Recommended Citation

Dissanayake, Geethmi Nimantha, "A Study on Real-Time Database Technology and Its Applications" (2020). *Masters Theses*. 4822.
https://thekeep.eiu.edu/theses/4822

A Study on Real-Time Database Technology and Its Applications

Geethmi Nimantha Dissanayake

Eastern Illinois University

Dr. Peter Ping Liu

Dr. Jerry Cloward

Dr. Wutthigrai Boonsuk

Table of Contents

Table of figures

1.Introduction

The advancement of popular technological appliances such as mobile phones and GPS trackers have increased the demand for real-time applications. These applications usually require up-to-the-second information by the users as well as the large volume of data to be processed along the speed of the actual application. These computing systems must react within precise time constraints while responding to the events in the operating environment (Buttazzo, 1997). The databases for these applications are expected to facilitate instant and massive data storage, instant access and delivery.

Cloud computing is a powerful technology to perform massive-scale and complex computing. It can be a part of the solutions where a massive amount of data and instant access are required such as in mobile computing applications. It eliminates the need to maintain expensive computing hardware, dedicated space, and software. Massive growth in the scale of data or big data generated through cloud computing has been observed (Hashem, et al., 2015).

Big data technology and the real-time databases were introduced as a solution in the emergence of larger and complex data sets, to offer better solutions and features to handle massive data sets with higher performance. Oracle defines big data as data that contains greater variety arriving in increasing volumes and with ever-higher velocity (Oracle Corp.). It helps organizations to integrate, manage and analyze data originated from various sources. Use of big data technology can be highly beneficial for a business, because of its ability to analyze the behaviors inside an organization, its actors and use those analyses to make important predictions. The traditional relational database model was no longer capable of fulfilling all above requirements.

Thus, a considerable portion of database architecture is transformed into a new form, from relational to real-time databases. Relational databases like MySQL store their data in tables, using rows and columns. The real-time databases like Firebase by Google, MongoDB use different database architectures like JavaScript Object Notation (JSON), or key-value. Unlike the traditional database model, real-time databases do real-time data streaming. Real-time data streaming uses multiple queries that work on time and buffer windows, making use of data while they are being processed in the server.

Real-time or ‘NoSQL’ databases are found to be useful when there are large sets of distributed data. The term NoSQL was first used to identify the databases built for large-scale database clustering in cloud and web applications. It is an approach to database design that can accommodate a wide variety of data. In contrast, traditional database management systems have data placed in a table structure and data schema is carefully designed before the database is built (Rouse, Vaughan, Rouse, Rouse, & Barney Beal, n.d.).

This research focuses on the foundation of real-time databases including their capability, key technology such as data storage mechanisms and performance, in relation to relational database commonly used in the field. A case study of a ride share application will be used to illustrate the major applications of real-time database.

1.1 Purpose

Many applications especially those associated with mobile devices are accompanied by the generation, storage, retrieval and processing of a large amount of data within a limited time. Due to the large volume of instant data processing, traditional relational databases cannot meet the challenge. A new type of database, known as ‘real-time database’, has been developed to meet the demand. This thesis presents a systematic study of real-time databases available to date.

1.2 Statement of Problem

Inside a large database system, a dataset can be subject to transactions in several times using various queries, e.g., traffic data of a specific route. The speed of data delivery and processing in this regard is expected to be increased efficiently with the use of a real-time database or a combination of a real-time database and a relational database. Other than speed, storage, retrieval, management or sorting are factors which come to the picture upon selecting a real time platform to manage data.

The rapid growth of data stream forces organizations to search for better solutions to store and manage a large volume of data. A real-time database management system has the ability to store, categorize and process these amounts of data. This ability helps the data processing and analytics applications to respond to customer queries quickly and easily. This helps the database administrators and management of a business to ‘be-prepared’ to face constant changes in information with speed and efficiency.

The variety of data which a business is going to receive and manage is becoming hardly predictable with the changing consumer requirements. Thus, the variety of data is one more challenge faced by the companies and the use of real-time databases can aid in such scenarios where data can be structured or unstructured or even hierarchical. Overall, real-time databases help the management face the challenges occurred by the characteristics of data like the volume, variety and velocity.

1.3 Limitation

To evaluate a specific real-time database, it should be installed in a working environment. Each type of real-time database has a specific set of requirements to run in a working environment. In this study, features of the most popular and available real-time databases like Firebase and MongoDB will be analyzed and discussed. They will be compared with the basic features of the traditional relational database management system. Services like Apache Kafka, Amazon Kinesis Data Streams have the ability of being real-time streaming platforms. In this study, the streaming platform will be limited to Apache.

Most of the real-time applications which are being used today are mobile, which means that they run on your smartphone. Mobile phone operating systems like Android and iOS have the ability to facilitate operations of these real-time applications. The features of the application might have some limitations depending upon the features of the mobile device in which it is being used. Each application might perform differently in different mobile phone operating systems. Therefore, the evaluations of this study may vary with the mobile phone operating system.

1.4 Delimitation

For this study, most available and used real-time database products like Firebase and MongoDB will be studied, discussed and the key features of them will be compared with each other and with those of the traditional relational database systems.

Among the available real-time databases, some have already become predominant. Firebase is a Google owned web and mobile application development platform, which provides a real-time database as a service. Several JavaScript frameworks such as AngularJS (MIT) and

Backbone.js (MIT) facilitate database access for Firebase real-time database. It is also accessible through a REST API: Representational State Transfer Application Program Interface which uses HTTP requests to GET, PUT, POST and DELETE data. Having a considerable number of worldwide users by today, Firebase real-time database has become one of the leading real-time database products.

MongoDB is a real-time database developed by MongoDB Inc., which is used in for high volume data storage. It is embedded with several features like indexing and file storage. It also has multiple editions and several architectures (MongoDB Inc.). According to these characteristics, MongoDB has become a real-time database product with a good market.

There are various operating systems which are being used in mobile and other computing devices. With the emerging technology, unlike the traditional real-time systems, modern real-time applications are involved with resource requirements like secondary storage devices and networks. This arises the need of operating system support in real-time disk access for real-time applications. Various operating systems would treat real-time database applications in various manners. The analysis regarding this study will be carried out in Windows operating system environment with the available resources. Thereby, the features of real-time databases discussed will be delimited to those which are operable in Windows environment only.

2. Literature Review

Real-time data and big data can play important roles in business decision making to help companies gain a competitive advantage by meeting ever increasing customer needs. Several studies have been conducted to investigate the impact of real-time databases and big data, on the corporate world. Yet very few studies have compared the features and feasibility of the existing real-time database products. This chapter will provide an overview of the existing literature on real-time database and applications, their performance and some previous studies in related areas.

2.1 Evolution of real-time databases

In most of the applications we have been using over several decades, relational database management systems are used to manage transactions. They were able to maintain data integrity, provide concurrent access, and facilitate data storage over this period. Arguably, these databases were able to perform well in almost all types of applications under different environments. However, the performance expectations from a database have been evolving over the time and the complexity of user requirements like amount of data these servers must handle, speed, storage, retrieval, management or sorting has been growing over time. In 2017, Microsoft SQL Server stated that the maximum size of database it could handle was 524,272 terabytes.

As a solution for the exponential growth of data and managing data integrity parallel to that growth, non-relational real-time databases were introduced. According to Strauch (2011), in most of the literature, a real-time database is defined as a database, where its transactions are defined considering the time of the validity of a data unit, the volume of data to be stored, the storage mechanism and the availability of the data.

According to Bansal and Chauhan (2017), a real-time database is advantageous over a relational database in several ways. A real-time database is more flexible than a traditional relational database, being schema-less. Also, many real-time database products allow sharding, which allows the users to add more machines and scale the cluster. These real-time databases support multiple storage engines. Therefore, unlike most traditional relational databases, the real-time databases have a variety of services which can facilitate the ultimate user requirements.

2.2 Early misconceptions about real-time databases

Stankovic, Son, & Hansson (1999) provided some examples for real-time databases and common misconceptions about the real-time aspects of databases. Several characteristics like transaction timing constraints, time semantics, correctness, and time consistency can be used to describe real-time databases to a user. The importance of these characteristics can be applied using real world scenarios like completion deadlines, stock market prices or sensor data (Stankovic, Son, & Hansson, 1999). Stankovic, Son, & Hansson (1999) find some confusions about real-time databases, which are based upon speed, database technology, and some real-time database properties: temporality, predictability and specialization.

For several decades, application developers and business managers were not well exposed to the real-time requirements of the consumers. Therefore, the difficulty of fulfilling real-time requirements with the traditional database systems were well hidden. As the amount of data in a database increases, it becomes difficult for the hardware to manage data. The accuracy of the transactions involving obsolete data cannot be guaranteed (Stankovic, Son, & Hansson, 1999). It was argued that the database being in the main memory can provide faster data access, but transaction scheduling and handling external interrupts cannot be effectively done. The users were unable to recognize that these challenges are difficult to overcome only with advancement of the existing database technology and hardware. Traditional database systems cannot be used for real-time applications simply by adding a few functional improvements (Stankovic, Son, & Hansson, 1999). Stankovic, Son, & Hansson (1999) mention that real-time databases are sometimes mistaken for temporal databases. Temporal databases are built for providing application-independent database management system (DBMS) support for time-varying information. They can have some fixed timescales (seconds or milliseconds) and storing the changes in the formatted data, which looks like a time series of data. A relational DBMS stores changes on data over time, using a timestamp, which is a discretely stored value for each measurement. Real-time databases are commonly optimized to act as time-constrained databases. They can be much more efficient than temporal or relational databases, in real-time computing applications that require timely access to data.

2.3 Types of Storage Models in Real-time Databases

According to the data handling methodology, the available real-time databases are divided into four major categories.

2.3.1 Key-value based store

Key-value is the simplest real-time database model in use. Data are stored as pairs of key-value, without any column-type relation. Figure 1 shows an example of such a database model using key-value based store. The keys of “Student ID” and “Course ID” are respectively associated with the values of “Student object” and “Course object”. The “Student object” may include more detailed information about the student, such as student name and address.

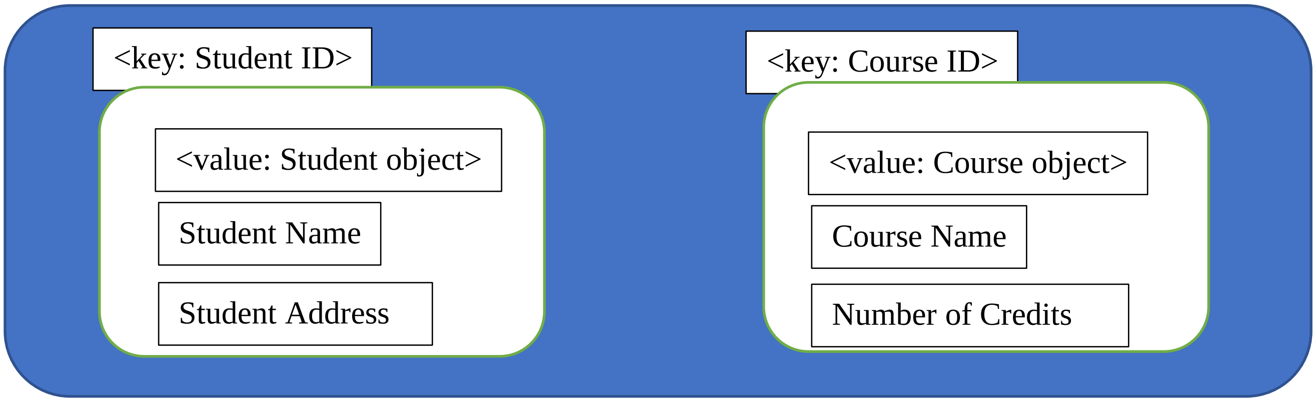


Figure 1. Illustration of a key-value model

JSON (JavaScript Object Notation) is the format of key-value objects. According to FileInfo(n.d.), JSON is a standard data interchange format, which is primarily used for transmitting data between a web application and a server. These are lightweight, text-based, human-readable files that can be edited using a text editor. The values may be in the form of any data type, simple text strings or binary objects. It has no structure or limitation. An application has to know what is stored and access a value with the key associated to it. A search for data or a query can generally only be implemented against keys, not values. These searches will attempt to find exact matches for the query. This model can be used for many web applications like user blogs, sessions, and ecommerce sites, as well as networking applications like telephone directories.

According to stackshare (n.d), services and companies like Instacart, Twitch, and Stack use Firebase, a key-value based store. It has built in libraries to support those above applications. Key-value real time databases are able to synchronize data between devices, which is needed by the above mentioned applications.

2.3.2 Document based store

In this real-time database model, data is stored as a value and it is associated with a unique identifier key. But unlike the key-value databases, document databases contain a structured or semi-structured value, which is referred as a document. A document store has many documents, and each document has a record and data associated with it. Figure 2 shows an example of such a database model. Document 1 is the root object and it has two other objects; Document 2 and Document 3 attached to it. Document 2 and Document 3 are respectively parents to Document 4, Document 5 and Document 6. Any of the other objects can be accessed via Document 1. But Document 4 and Document 5 can be accessed only using Document 1 and Document 2. Similarly Document 6 cannot be accessed via Document 2, but Document 1 and Document 3.

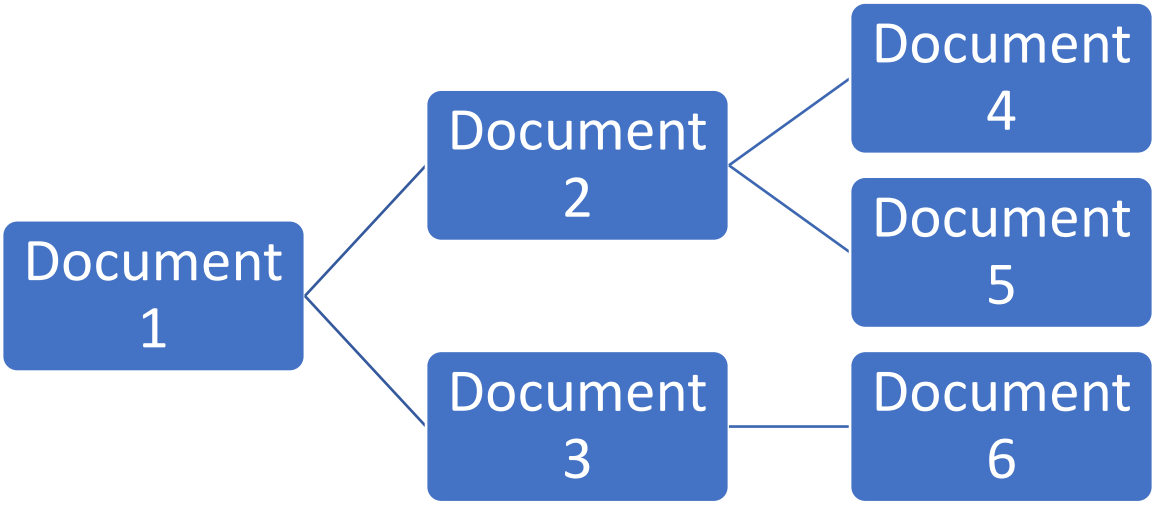


Figure 2. Illustration of a document based database model

The data inside a document is semi-structured. The data can be encoded using several methods, such as JSON (JavaScript Object Notation), BSON (Binary JSON), binary, etc. A document database can be easily queried using many languages like JavaScript, Python, Java, etc. This type of real-time database is suitable for web applications, catalogs, gaming sites, and networking applications. The simplest real-world example one can use to describe this structure is a ‘tree’. A tree has a root, branches, and other branches attached to those branches. A document-based store stores data in the same hierarchy. For every key, a record(document) can be stored as value. Further, these documents have their own internal structure.

These documents based real-time databases were specifically designed for the use in web services. Thus, most common web technologies such as JavaScript and HTTP facilitate the integration of document-based databases (Meier & Kaufmann, 2019). These come in handy in processing large amounts of data in web sites, because of their horizontal scalability supporting storage of a large volume of data. The data can be in a wide variety of types like text, worksheets, emails, XML documents and many more.

2.3.3 Column based store

This model stores data using a column-oriented model. Column based real-time databases are sometimes referred as wide-column or columnar stores. Data is stored in columns rather than rows. Figure 3 shows an example of such a database model. The database “Student Profile” contains two records for two students, Gregory and Wolowitz. Each record has data values for the columns ‘email’ and ‘major’ of the student. But only Gregory has provided his ‘age’. Each of these values inside columns can be identified with the unique identifier for eah student; ‘Student ID’.

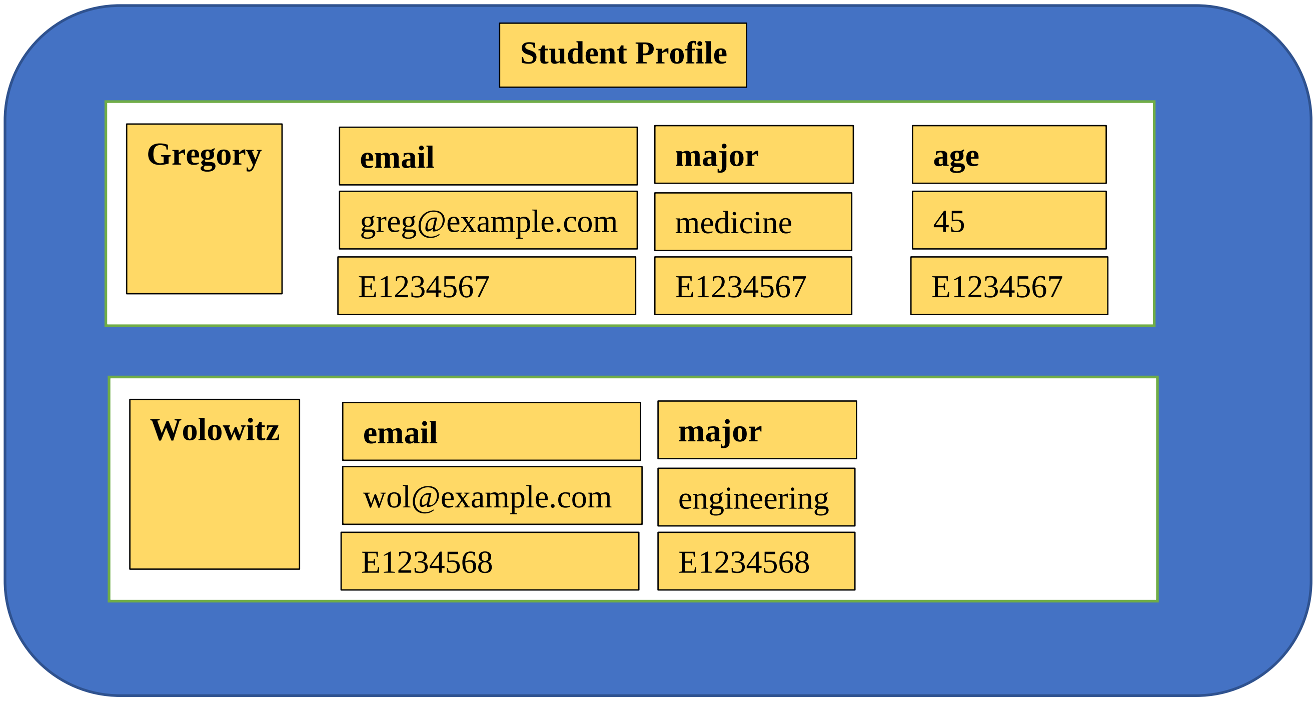


Figure 3. Illustration of a column-based database model

Similar to the schema of a relational database, a column-based store uses a “keyspace.” A keyspace has column families, which are used to group data. A column family has one or more rows, having different columns. A single column is a tuple with a unique name, value, and timestamp, and it is contained to its row.

2.3.4 Graph based store

A graph-based store uses a graphical model to store data. It uses a graphical structure with nodes, edges, and properties. This model has no predefined schema for the graph. The schema is built up according to the data entered, and with the addition of more data, the schema grows. This type of real-time database can support interconnected data. Figure 4 shows an example of such a database model. The entity ‘Facebook Status of Chen: Married’ has different relationships with the other entities of the database. The entities inside the database belong to different types; people, statement, location etc. Any entity can have a relationship with their entities. As an example, ‘Monica’ and ‘Cynthia’, who are both related with the ‘Status’ object are already related as ‘friends’.

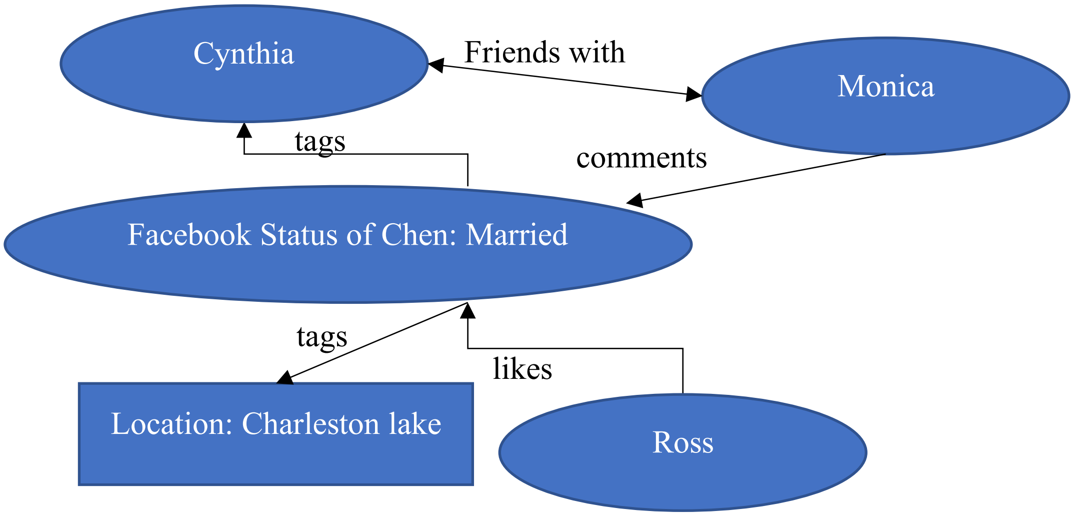


Figure 4. Illustration of a graph database model

2.4 Performance Evaluation of Real-Time Databases

Wang, et al. (2015) has proposed a model to analyze data structure, architecture, volume, concurrency, availability and scalability of different database. The proposed model was used to evaluate the Velocity, Volume and Variety of the databases, which are important metrics in real-time applications. The study has focused in-memory databases, which use memory storage of a computer, instead of disk storage. Wang, et al. (2015) have evaluated whether an in-memory database could accelerate the real-time transaction in an efficient way. This study also analyzes and discusses the design and optimization of real-time processing techniques to improve the performance of memory database. The importance of analysis and optimization methods for enhancing the processing speed of the financial, securities and other industries' critical trading applications is also pointed out by the authors (Wang, et al., 2015). Therefore, it is noted that evaluating and analyzing the capabilities of different databases has been identified as a need for the corporate world.

MySQL is arguably the most widely used relational database management system (Roopak, Rao, Ritesh, & Chickerur, 2013). Roopak, Rao, Ritesh, & Chickerur (2013) present a performance analysis between relational database MySQL and object database db4o based upon Huge Airport database. db4o is an open source object database, which is available for .NET and Java. The study is performed by executing various queries for insert, select and delete operations on the same data set using SQL and db4o databases. The paper concludes the study by showing that db4o is found to be more efficient on insert operations, but less efficient on delete and select operations (Roopak, Rao, Ritesh, & Chickerur, 2013).

Kumar, Srividya, & Mohanavalli (2017) have proposed the performance comparison of two document-based real-time databases, MongoDb and CouchDB. This is a qualitative as well as a quantitative study, focusing a streamline application; twitter. Real-time databases can be helpful in some applications, where a traditional database cannot be applied.

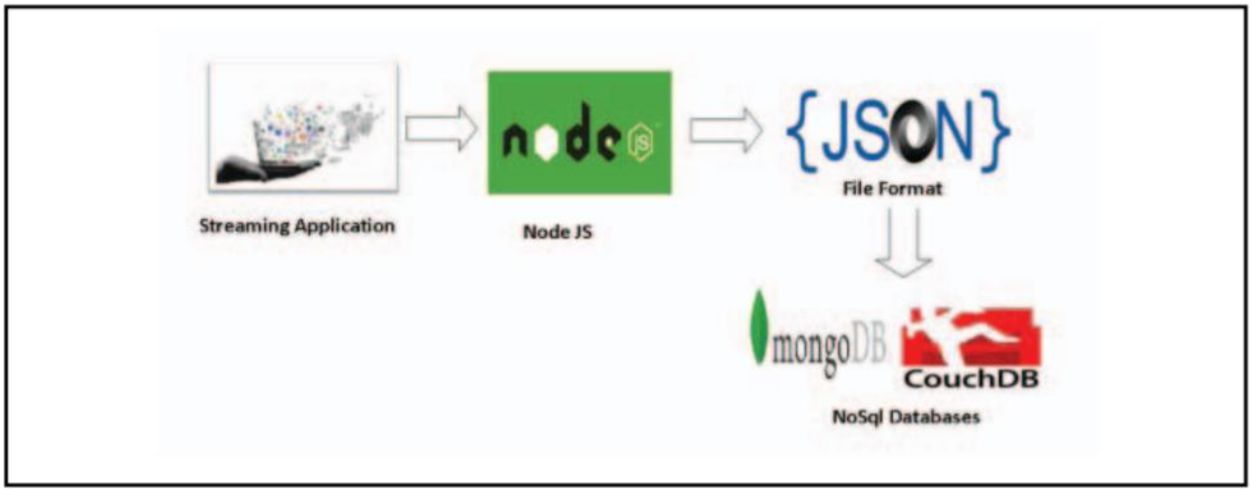


Figure 5.Proposed database comparison framework (Kumar, Srividya, & Mohanavalli 2017)

Figure 5 shows the proposed framework for selecting the appropriate database, which runs on a platform having NodeJS and JSON, which are relatively a runtime environment for streamline applications, and a format for storing and transporting data (Kumar, Srividya, & Mohanavalli, 2017). The Node.js server accept a request from the application and calls a function to retrieve the requested file from disk. While waiting for that file, it processes the next request. After selecting an appropriate server, the data format is taken into consideration. Here, the format is shown as JSON, which is object oriented. Finally, the suitable real-time database for the chosen server and file format is selected, among the existing real-time database products. They have used CouchDB and MongoDB, two document-oriented databases for this study, and they have concluded that MongoDB performs well in the considered applications. For the evaluation of results, Kumar, Srividya, & Mohanavalli (2017) have considered factors like the size of the data which can be stored in a database, performance of the database when various types of queries are encountered, and bulk-importing of JSON documents. Further, a performance comparison of MongoDB and OrientDB also has been performed evaluating the insert, update, delete, and search operations (Tavares, Oliveira, & Lóscio, 2016).

2.5 Trending Usage and Popularity of Real-time databases

The better way to gain an insight on current real-time database usage, their popularity and important features to consider is surveying the opinions of the information technology community. The most common metric tracked for evaluating the performance of a database happens to be query response time. Figure 6 shows how the features which are taken into consideration upon tracking the performance of a database, are dispersed. Characteristics like response time, memory, system reliability and other features are taken into consideration for this comparison.

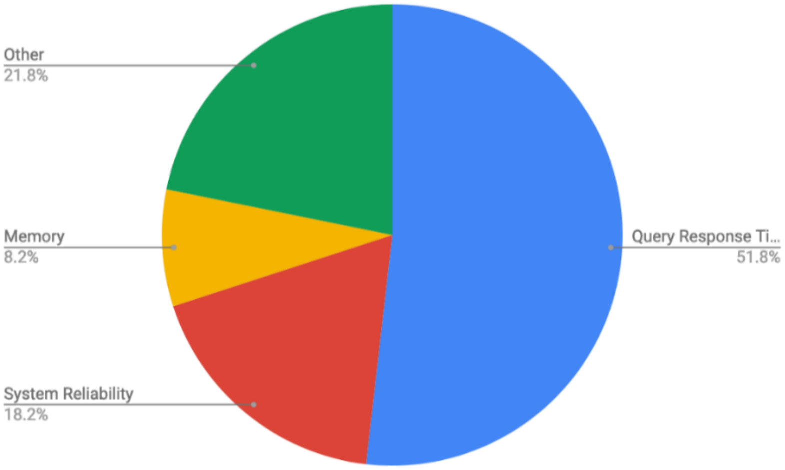


Figure 6. Most important metrics tracked for database performance (2019)

It is found that majority of the characteristics expected from a database consists of efficient response time and reliability.

A survey was carried out by DeveloperWeek (2019). According to the results, the usage of relational database is 60% among the participants. Among the available databases, an evaluation of the popularity can give a brief idea about the reliability and accessibility of them.

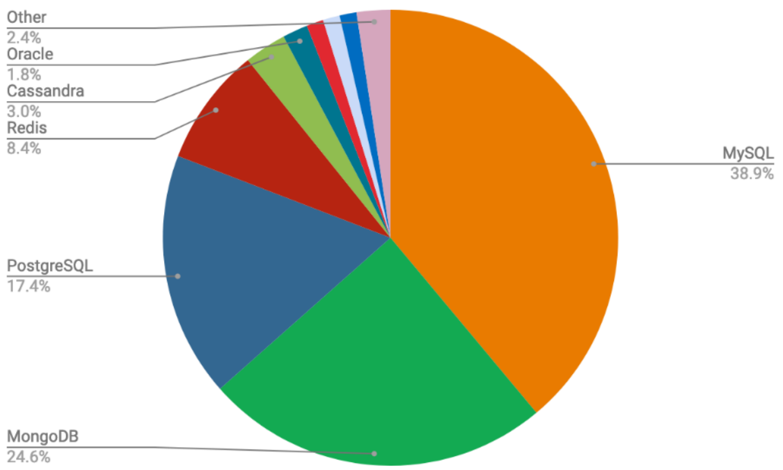


Figure 7. Most popular databases (2019)

Figure 7 depicts that still the dominance of MySQL has not been shaded yet. It is to be noted that 'Other' category in this chart contains a combination of CouchDB, Berkeley DB, Microsoft SQL Server, Redshift, Firebase, Elasticsearch, and InfluxDB users. It is visible that MongoDB real-time database has a notable popularity among all the relational and real-time databases. DB-Engines Ranking has also ranked database management systems according to their popularity.

Figure 8 displays the results of this evaluation**.** The top three curves represent Oracle, MySQL, and MSSQL database engines, which operate as relational databases. Apart from their consistency in the rankings, it can be seen that the curve which represents MongoDB has a significant growth, as well as the one which represents Redis real-time database management system. Apart from the steady popularity of the relational database engines like Oracle, MySQL, Microsoft SQL, it can be seen that the latest real time database technologies have been gaining popularity over the time.

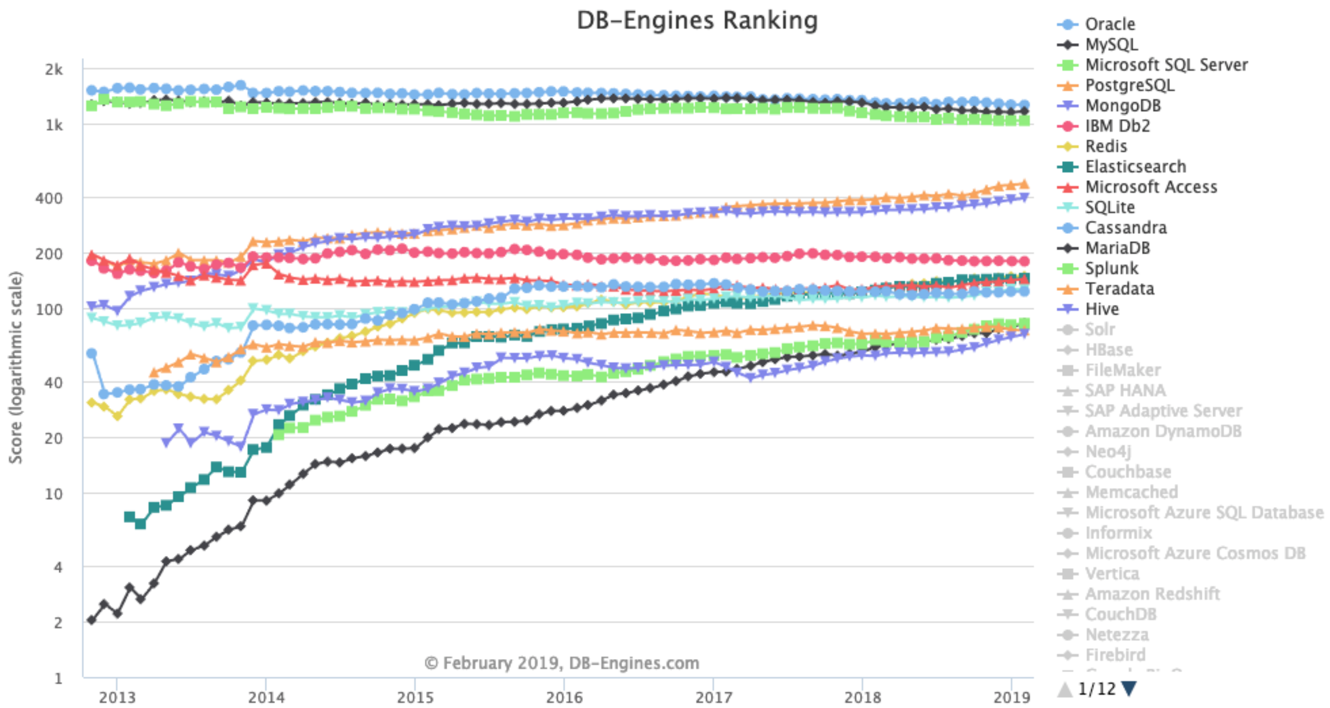


Figure 8. Database engines ranking (2019)

The growth of the volume of data in existing relational SQL databases and the advancement of real-time database applications have bent the developers' attention towards the use of multiple database combinations. Arguably, both relational and non-relational databases have their own advantages upon each type of application. The survey by DeveloperWeek (2019) extends to tabulate the participants responses about such applications.

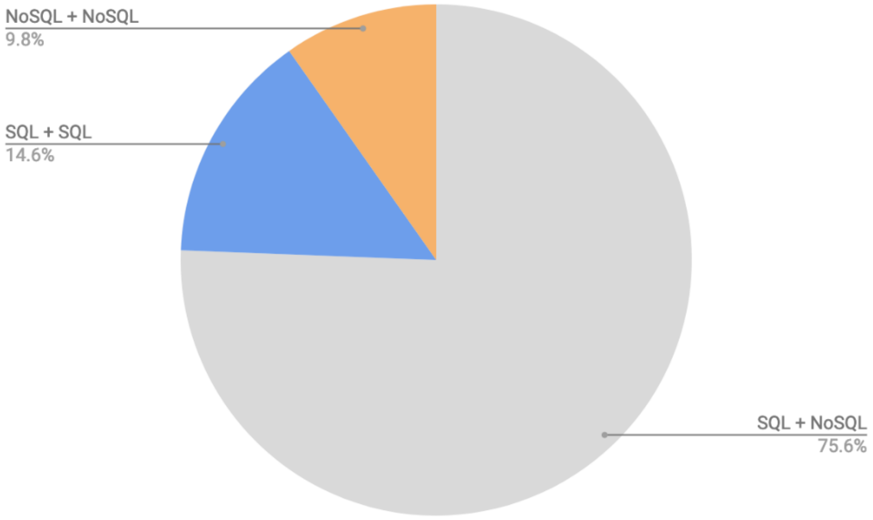


Figure 9. SQL & NoSQL Multiple Database Combinations (2019)

Figure 9 shows that there is a high probability of using SQL and a real-time databases in combination. The combined use of SQL+real-time database has a high popularity. These findings solidify the fact that an organizational could have real-time as well as non-real-time requirements, therefore it is to be taken into consideration that fulfilling these needs are being done with the use multiple database combinations.

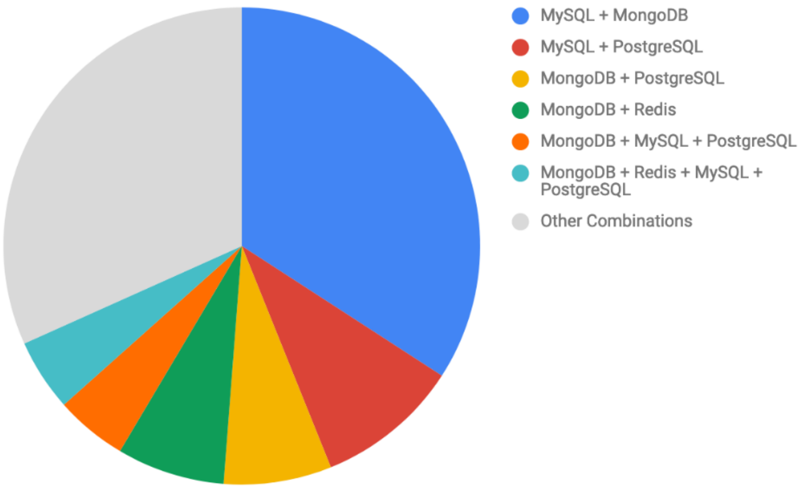


Figure 10. Most popular multiple database type combinations (2019)

Among the combinations of databases used by the participants, MySQL and MongoDB combination comprises a notable portion of the chart shown in Figure 10, which accounts 34%. It is the highest among all represented database combinations.

2.6 Ridesharing concept

The casual 'carpooling' approach, which is based on meeting places is considered as the seed for invention of the ride sharing concept. The database of the Web of Science currently includes 289 articles, which have content written about ridesharing and its economical aspects (Ključnikov, Popesko, & Kloudová, 2019). Ride-sharing has been regarded as an effective way to reduce carbon emissions and alleviate urban transportation congestion (Wang, Wang, Wang, Wei, & Wang, 2018). Another study shows that it has emerged as a solution for major transportation problems, such as high gas cost, finite oil supplies, and the time-consuming traffic, which is environmentally and socially sustainable (Xing, Warden, Nicolai, & Herzog, 2009). Since the late 1990s, numerous ridesharing applications have integrated the Internet, mobile phones, and social networking into their services. Real-time ride sharing services employ “smartphones”, automated ride sharing software, GPS and wireless networks. Therefore, dynamic ride sharing has and will gain more popularity in the coming years.

Furuhata, et al.(2013) have stated that ridesharing takes on different characteristics, thus different forms. Figure 11 shows four such patterns which were identified in ridesharing services in this study. Those are identical(both the origin and destination of a driver and a passenger are identical), inclusive(both the origin and destination of a passenger is on the way of an original route of a driver), partial(both the pick-up location and drop-off location of a passenger are on the way of an original route of a driver, but either the origin or the destination of the passenger is not on the way) and detour(either the pick-up location or drop-off location or both of a passenger are not on the way of an original route of a driver. Taking a detour, ridesharing route covers both the pick-up and dropoff locations). The researchers have compared those patterns with reference to single passenger booking a ride vs multiple passengers sharing a ride.

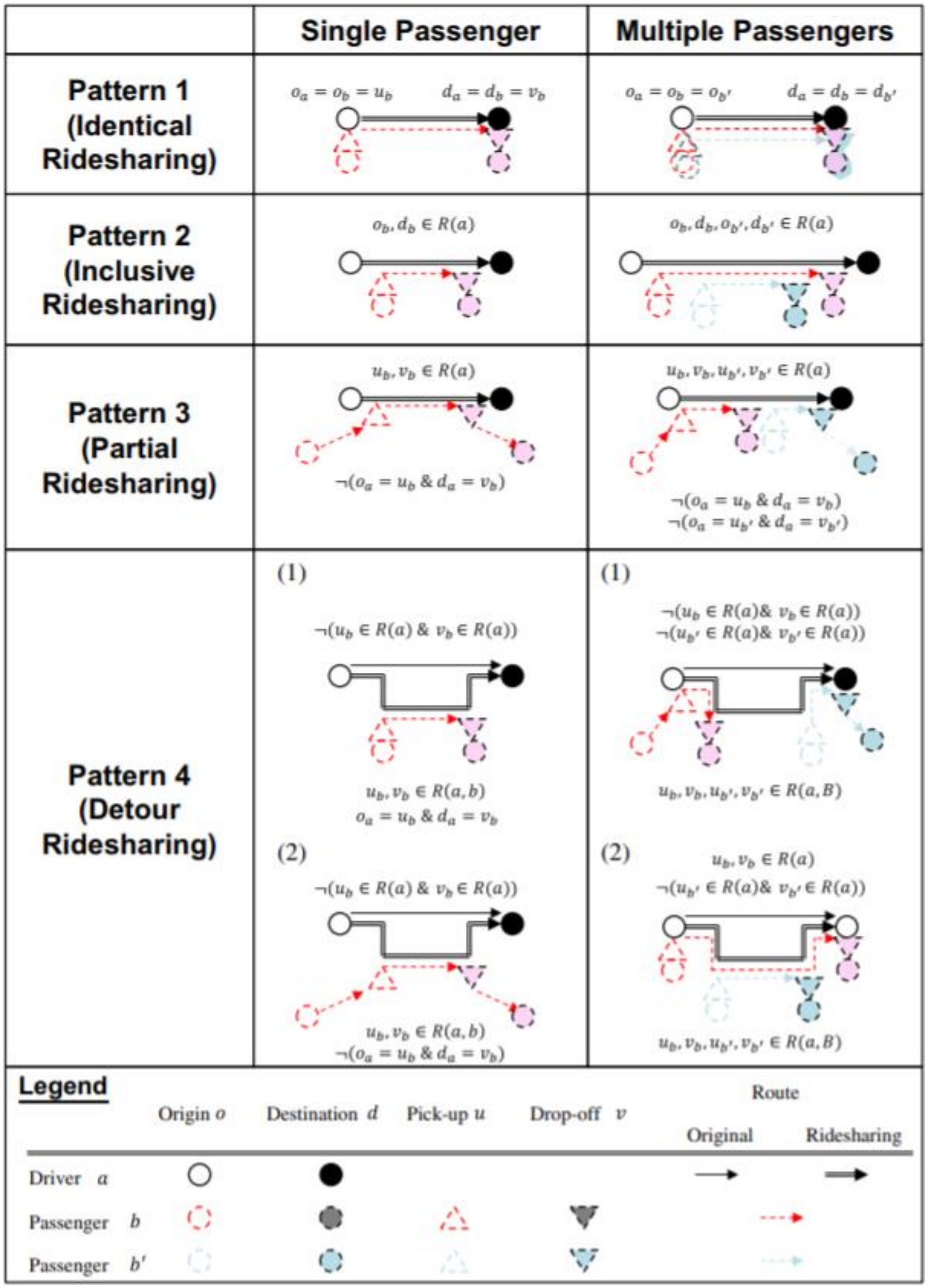

Single Passenger
Multiple Passengers
Pattern 1 (Identical Ridesharing)
$o_a = o_b = u_b$
$d_a = d_b = v_b$
$o_a = o_b = o_{b'}$
$d_a = d_b = d_{b'}$
Pattern 2 (Inclusive Ridesharing)
$o_b, d_b \in R(a)$
$o_b, d_b, o_{b'}, d_{b'} \in R(a)$
Pattern 3 (Partial Ridesharing)
$u_b, v_b \in R(a)$
$\neg(o_a = u_b \,\&\, d_a = v_b)$
$u_b, v_b, u_{b'}, v_{b'} \in R(a)$
$\neg(o_a = u_b \,\&\, d_a = v_b)$
$\neg(o_a = u_{b'} \,\&\, d_a = v_{b'})$
Pattern 4 (Detour Ridesharing)
(1)
$\neg(u_b \in R(a) \,\&\, v_b \in R(a))$
$u_b, v_b \in R(a, b)$
$o_a = u_b \,\&\, d_a = v_b$
(1)
$\neg(u_b \in R(a) \& \, v_b \in R(a))$
$\neg(u_{b'} \in R(a) \& \, v_{b'} \in R(a))$
$u_b, v_b, u_{b'}, v_{b'} \in R(a, B)$
(2)
$\neg(u_b \in R(a) \,\&\, v_b \in R(a))$
$u_b, v_b \in R(a, b)$
$\neg(o_a = u_b \,\&\, d_a = v_b)$
(2)
$u_b, v_b \in R(a)$
$\neg(u_{b'} \in R(a) \& \, v_{b'} \in R(a))$
$u_b, v_b, u_{b'}, v_{b'} \in R(a, B)$
Legend
Origin o
Destination d
Pick-up u
Drop-off v
Route
Original
Ridesharing
Driver a
Passenger b
Passenger b'

Figure 11: Ridesharing patterns (Furuhata, et al.,2013)

Uber is a ridesharing app, that allows consumers to order a private or shared car with the aid of mobile technology and online payment/banking options. It is proven to be more convenient than hailing a traditional cab. Following figure 12 depicts how the popularity of traditional transportation systems have faded away with time. This shows a comparison of the popularity between ground transportation mediums like rental cars, ride hailing and taxis from 2014-2018, As the graph displays, the revolutionary ride-hailing services have gained much popularity over the time.

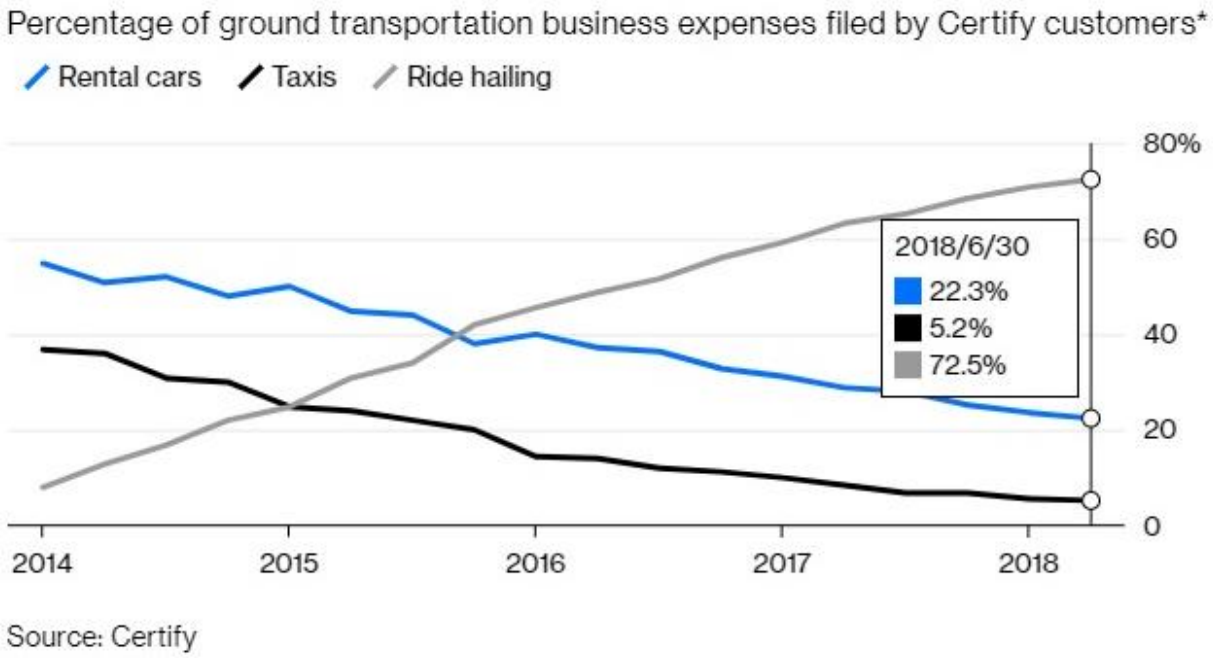


Figure 12: Ride hailing popularity (Iqbal, 2020)

Figure 13 shows the growth of usage of Uber over the years 2012-2018 (Brickell, 2019). It shows that the number of trips gone by Uber drivers has reached 10 billion by September 2018. This is a significant advance comparing to 2012.

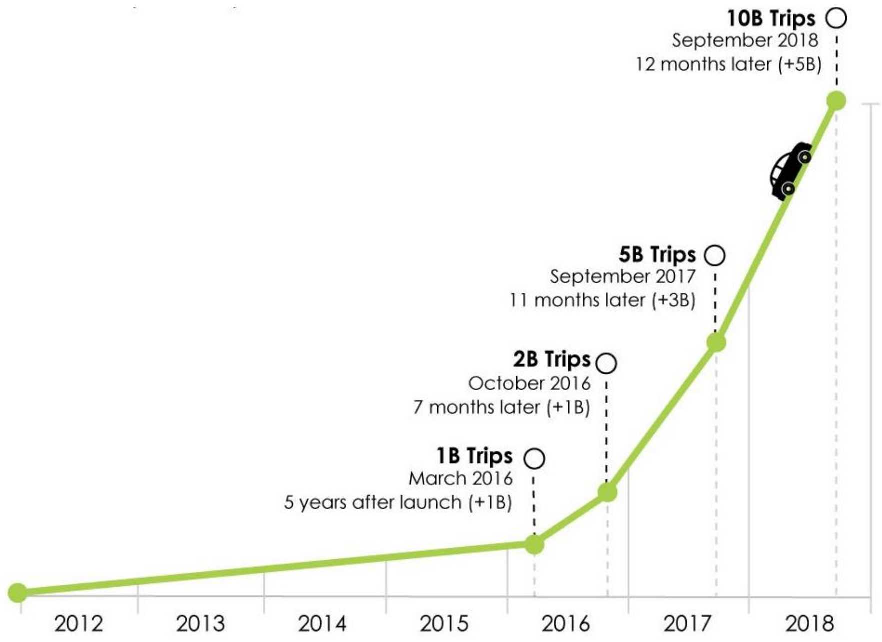


Figure 13. Uber's hypergrowth (Brickell, 2019)

Ride share platforms are most of the times aligned with smartphone applications which keep the drivers connected to passengers in an area. The use of sophisticated technological aspects has made it possible for the consumers to fulfill their requirements with one tap on the smartphones. These features altogether have made the ridesharing experience more attractive to the passengers.

3. Research Methodology

The purpose of using a real-time database for an application is to facilitate time-constrained access to data that has temporal validity. It is important to gain an in-depth knowledge about how the existing real-time databases differ from one another and what are their strengths and weaknesses which can affect the performance of your application. The purpose of this research work is to identify the most important features associated with real-time databases, to generate an overview of this technology for the users to refer when selecting an appropriate real-time database. This chapter will describe the methodology used for this research work.

3.1 System Setup

- **Hardware**

  Intel(R) Core i5-7200U @ 2.50GHz personal computer with 8GB RAM and 120GB of hard disk space

- **Operating System Environment**

  Microsoft Windows 10 operating system

- **Other Software Used**

  MongoDB Compass Community

  Google Cloud Firebase

  Microsoft Visio

3.2 Design of the Study

This research work follows a qualitative method, which is carried out in three major phases. The first phase occupies a literature review of the existing works on rendering real-time databases, which are available in the market. The second phase will select four real-time database applications to represent the four types of real-time databases, which are available in the market, and compare their prominent features. The third phase continues with a case study of a ride share application. A database feasibility evaluation will be conducted considering the real-time demands of a ride share application.

3.2.1 Feature comparison

Database systems are the primary providers of business intelligence and enterprise logic which are the keys to success. To accomplish these, companies will be using high performance software systems. In parallel to those software systems, database management systems also need to be efficient in terms of speed and storage. Because the data of a business is an ingredient for decision making process, reliability, availability, maintenance cost and fault-tolerance are also huge concerns for a database system.

Previously mentioned features are considered in common for relational and real-time database management systems. Apart from those, we will identify some key features for both relational and real-time databases. Some examples of database features include storage type, querying, indexing, scalability and transactions. These features will be analyzed and discussed for each of the real-time database application we are investigating.

Four real-time database applications representing the four major types of real-time databases: key-value based, document based, column based and graph based, will be taken into consideration in this study for evaluating the key features of each type of real-time databases.

- Key-value based store

Google Firebase real-time database, which is a key-value based real-time database, will be used as a model database to analyze the features of a key-value based store. Firebase is highly available and easily installable and offers both free and paid options. It can be deployed in Windows and LINUX environments.

- Document based store

MongoDB real-time database, which is a document-based real-time database, will represent the document-based store in our study. MongoDB is widely used, is highly available and is easily installable. It can be deployed in Windows and LINUX environments.

- Column based store

Apache Cassandra is a column-based real-time database, developed by Apache Software Foundation. It is free and open-source. Cassandra was originally developed at Facebook, was open sourced in 2008, and became a top-level Apache project in 2010 (Shukla, 2016).

- Graph based store

OrientDB is an open source real-time database management system which supports multiple data store models, including graph-based store. Notably, the relationships are managed as in graph databases with direct connections between records. It is owned by OrientDB Ltd.

3.2.2 Case Study

The advent of smartphone has been able to introduce ride-sharing applications for taxi services. Among the many applications which use real-time databases for their data management, ride share and traffic applications are found in common. These applications process a considerable amount of data at a time, within their operations. Therefore, a ride share application will be used in this thesis, to evaluate the types of real-time databases and perform the necessary evaluations prior to selecting a database type for the application.

The database directly affects the operability of any ridesharing application. It should maintain up-to-minute (or up-to-second) the data about the routes, locations, user details and ride history. Real-time databases, unlike relational databases where modeling can be done using which has entity relationship and class diagrams, has no constraints for data modeling diagrams, allowing the database modeling to start with minimum requirements. Therefore, entity relationship modeling will be used to identify the minimum requirements.

3.2.3 Ride share application

As an application development conducted by Muhammad Rehaan Saeed of the School of Technology, Eastern Illinois University, a ride sharing application has been developed using NativeScript and Google Cloud Platform. NativeScript is a development framework for mobile applications such as iOS & Android, which can facilitate these technologies serving as a smartphone native application. Google Cloud Platform will provide Google Maps plugins, Google Places and Geolocation API for iOS & Android applications. This application was developed targeting Eastern Illinois University students and staff. The user authentication will use the Eastern Illinois University active directory data to verify login into the application. If there is any complaint on a ride, the authenticity of the users will be tracked by using the active directory data. This application was used in this study to evaluate the feasibility among the selected real-time database products by establishing a connection with between the application and the database, and performing basic create, read, update, delete operations on the application in the mobile environment.

4. Feature Comparison

The feature comparison of real-time database applications will be conducted on the four major types of real-time databases.

4.1 Google Firebase : A key-value based store

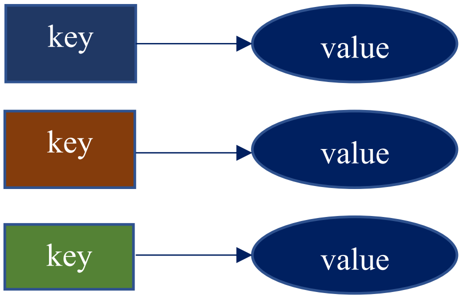


Figure 14: Key-value structure

Figure 14 represents the most basic structure of a key-value type real-time database. Google Firebase real-time database, which is a key-value based real-time database, will be used as a model database to analyze the features of a key-value based store. The following figure (Figure 15) represents a sample key-value type real-time database.

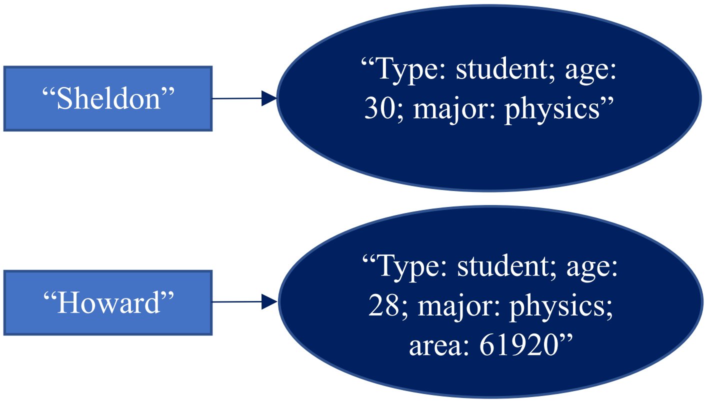

"Sheldon"
"Type: student; age: 30; major: physics"
"Howard"
"Type: student; age: 28; major: physics; area: 61920"

Figure 15: Key-value example

There are several reasons for selecting this specific database, Firebase as the model for a key-value type real-time database. Firebase is highly available and easily installable and offers both free and paid options. It can be deployed in Windows and LINUX environments. This cloud-based real-time database supports concurrent access for its clients in real-time, also providing offline functionality of the database (Moroney, 2017). The platform of Google Firebase makes using the Firebase database highly efficient. Multiple services such as data storage, hosting and validation are provided by the single Google Firebase platform, minimizing the need of gaining the services from other parties for the user or developer. This vast number of services in prevent the clients engaging with other service providers, keeping an upscale clientele in the business. Firebase provides features such as,

Storage : Firebase is backed by Google Cloud Storage which is a popular cost-effective data storage service, thus able to facilitate easy and secure file transfer regardless of network quality for the applications. It can store images, audio, video, or other user-generated content.

Hosting : Firebase can distribute web content in a fast and secured manner.

Authentication : Firebase lets the developer control and customize the access privileges of the users for the application. Firebase offers social login through providers like Gmail, Github, Twitter, and Facebook. It is a paid service that can authenticate users using only client-side code. The user management strategy also has an email and password verification process for the login.

Analytics : Firebase has a paid app measurement option that also provides user engagement, enabling the application developer to analyze the usage of the application. This tool captures events and properties on its own and also allows getting custom data.

Crash reporting : Firebase creates detailed error reports. Errors get listed into different groups on the basis of the harshness of the error. This aids in crash prevention to the developer.

4.2 MongoDB : A document-based store

MongoDB is considered to be the most popular document based real-time database in the market. It is open source and stores data as BSON(Binary JSON). It is document dependent and there is no schema in the database at all. Figure 16 represents the basic structure of a MongoDB database.

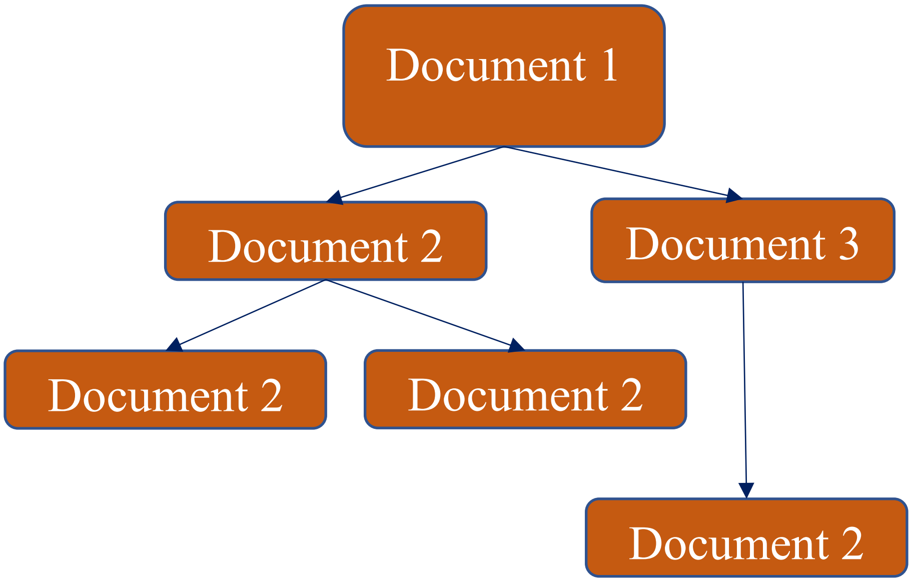


Figure 16: Document based store

The primary purpose of using MongoDB for managing data, is storing everything together at one place. The MongoDB database is merely a collection of documents. One document is like a row of a table in a structured database. Inside a document will be key-value pairs: a key-value pair has the same structure and characteristics of key-value based stores. The key-value pairs will be created until it runs out of instances and these pairs altogether will build up one document. Figure 17 shows an example of a document of a MongoDB database. It has three rows of data having values which are not identical. The rows contain student details, where the three students have different data values for their names and Id, but not for their occupation.

Id: 001, “name”: “sheldon”, “occupation”: “physicist”
Id: 231, “name”: “howard”, “occupation”: “physicist”, “age”
Id: 3443, “name”: “raj”, “occupation”: “physicist”

Figure 17: MongoDB example

A set of these documents will create a collection, which is the equivalent of the table in relational databases. A MongoDB database is made of multiple collections.

Companies like Adobe, ebay, cisco and SAP are popular examples of companies who have started using this product. The growth of MongoDB community is due to several features it offers to its users, being a real-time database. MongoDB is a schema-less database. Unlike a traditional SQL database, there are no complex joins when retrieving data from MongoDB. But it can facilitate deep querying because it supports a powerful dynamic query on documents. A MongoDB database instance uses internal memory to store data sets, thus provides fast access. Moreover, it is easily scalable. The application maintenance is made easier because of the horizontal scalability. This flexibility has eliminated the problem of schema redesigning when it comes to large commercial databases.

4.3 Cassandra: A column-based store

Cassandra real-time database is written in Java language, which was developed and now distributed by Apache Software Foundation. Cassandra was initially created at Facebook. It is a combination of the Amazon’s Dynamo storage system accompanied by Google’s Bigtable model. Currently, it is also a way of getting data from point to point for other databases such as ScyllaDB, Azure's CosmosDB and YugaByte, when multiple applications are involved. It is an open source database system and is designed to manage large transactional data across various server globally. It has an architecture without any master node to handle all the nodes in the ring or network. The decentralized data distribution within nodes in this architecture is in equal probation. Following Figure 18 shows the structure of the data storage Cassandra has. The records have different data values attached to them. Each color represents a single type of records or data, which can be names, images, addresses etc. The number of columns for a record is not identical in this format of real-time databases.

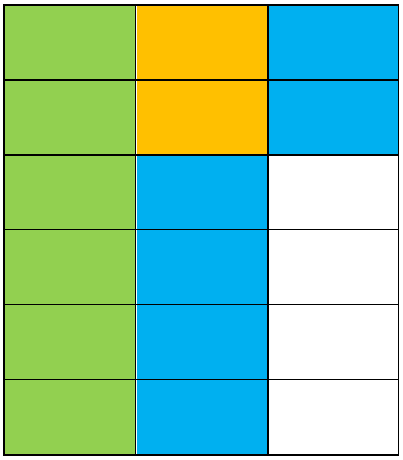
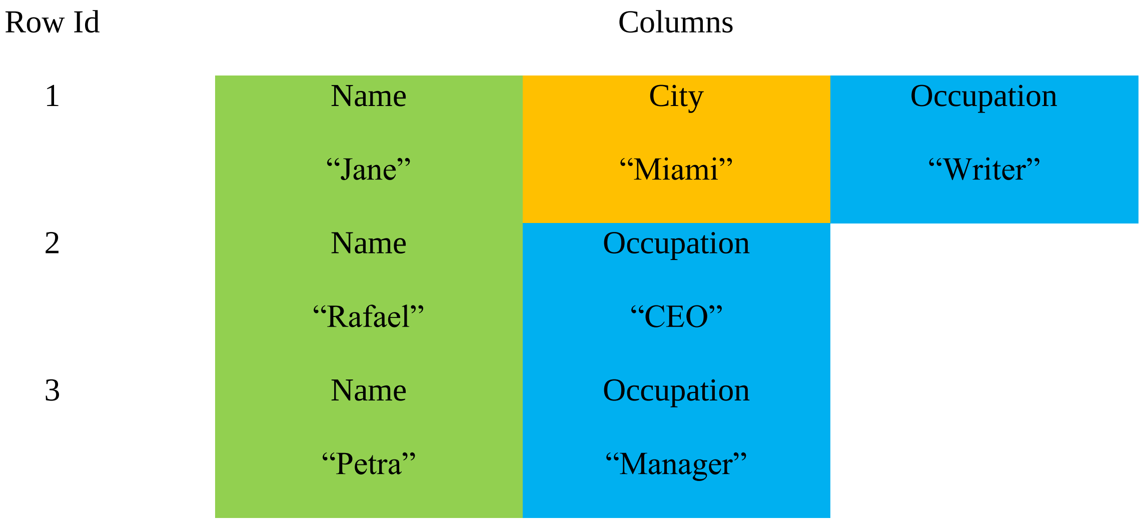


Figure 18: Column based model

Cassandra offers very high availability and very high fault tolerance with zero downtime. The decentralized data distribution keeps the performance unharmed even if an entire datacenter is lost. Cassandra offers the near-constant availability which required to support real-time querying for web and mobile applications. Cassandra architecture is capable of handling large datasets. Following figure displays an example of a simple Cassandra data store. . The records have maximum three different data values attached to them. One record has two data values and the rest of them have only two data values. Therefore, the number of columns for a record is not identical in this format of real-time databases.



Row Id	Columns		
1	Name “Jane”	City “Miami”	Occupation “Writer”
2	Name “Rafael”	Occupation “CEO”	
3	Name “Petra”	Occupation “Manager”	

Figure 19:Cassandra example

It offers wide-column flexibility as well as flexible parsing. Cassandra is easy to setup and maintain. Flexible parsing and wide column requirements allow applications to write into any node anywhere and anytime. Other than those, automatic workload management and data balancing across the nodes are also facilitated. In contrast with the Firebase or MongoDB, Cassandra is linearly scalable, supporting addition of more nodes to the cluster. It should be kept in mind that since Cassandra is a distributed database. Therefore, how the data and the workload will be distributed should be identified prior to designing the database and the data model. Thus, factors like size of the tables, partition size should be identified, when using Cassandra.

4.4 Orient DB : A graph based store

OrientDB is a multi-model open source real-time database management system that supports data models in documents, graphs, key-value, and objects. It is written in Java and was launched in 2011 by OrientDB Ltd. OrientDB is transactional and supports distributed architecture with replication. The manipulation of the database can be done in Java, SQL or with Gremlin. Physical data storage can be done in memory and on disk. Like all systems, it uses the free adjacency list to enable native query processing.

5. Case Study

Typically, a ride share system contains two applications, a driver side application and a user side application. Ride share platforms use a smartphone application which connects drivers with passengers in the area. The driver must log into the application and confirm his availability to accept a ride. Figure 20 illustrates the basic modules of a ride share application, driver, passengers, and vehicle.

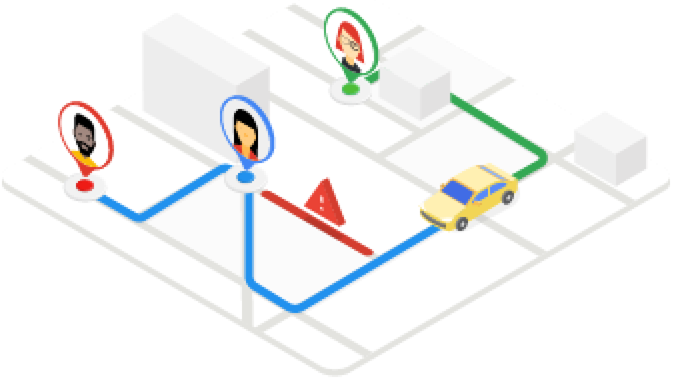


Figure 20: Ridesharing

Several specifications were gathered for a typical ride share application and the following were identified as the key features.

- Cross-platform mobile application
- Authentication for different user roles
- MapView for passengers and drivers to share the location
- Passenger functionalities
- Driver functionalities

Some pre-requisites are to be considered here, before designing a database model. To be registered as a driver, one must have a valid driving license. For the ridesharing service to be used, both parties (driver and passenger) must have the ride share application installed in their smartphones. Figure 21 illustrates the basic flows among the entities of a ride sharing scenario.

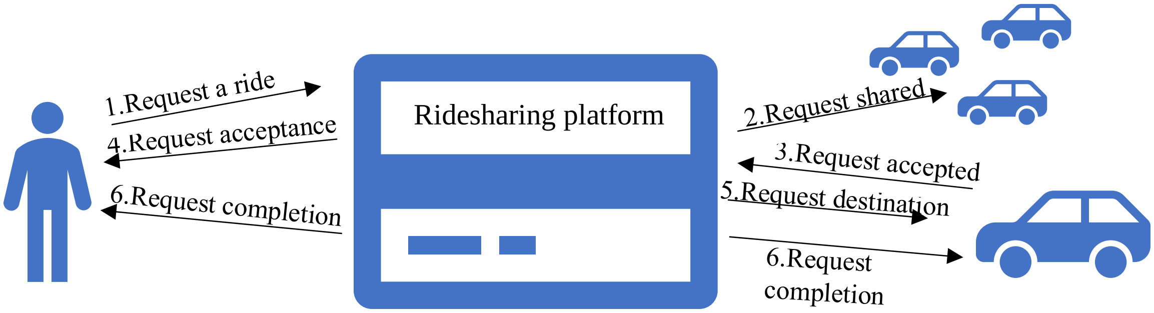


Figure 21: Rideshare flow

According to the provided scenario, obvious and implied entities for a ride share would be as following.

Ride request: The passenger logs into the application and pins his location. Once a passenger selects a destination and requests a ride, the application will send notifications to the nearest available drivers. The driver who accepts the request will drive to the passenger’s location using an integrated GPS service.

Ride: Once the ride has been accepted by a driver and the passenger has been picked up, the driver proceeds to the passenger’s destination. He will then pick the passenger up, update the status of the ride as ‘started’. The mobile GPS service tracks the ride throughout the route.

Arrive: When the driver reaches the requested destination, the trip will be completed, and the passenger will exit the vehicle. Here, the ride is completed, and the passenger can update the application and provide feedback if preferred.

The database for a ridesharing application should store data in separate modules such as Rides, Users and Requests. With reference to the first design iteration, a basic diagram could be improved as the application development proceeds.

Unlike SQL, real-time data modeling lacks conventional names and design principles. An entity-relation (ER) model is used to identify the real-world entities and the relationship between them in this context.

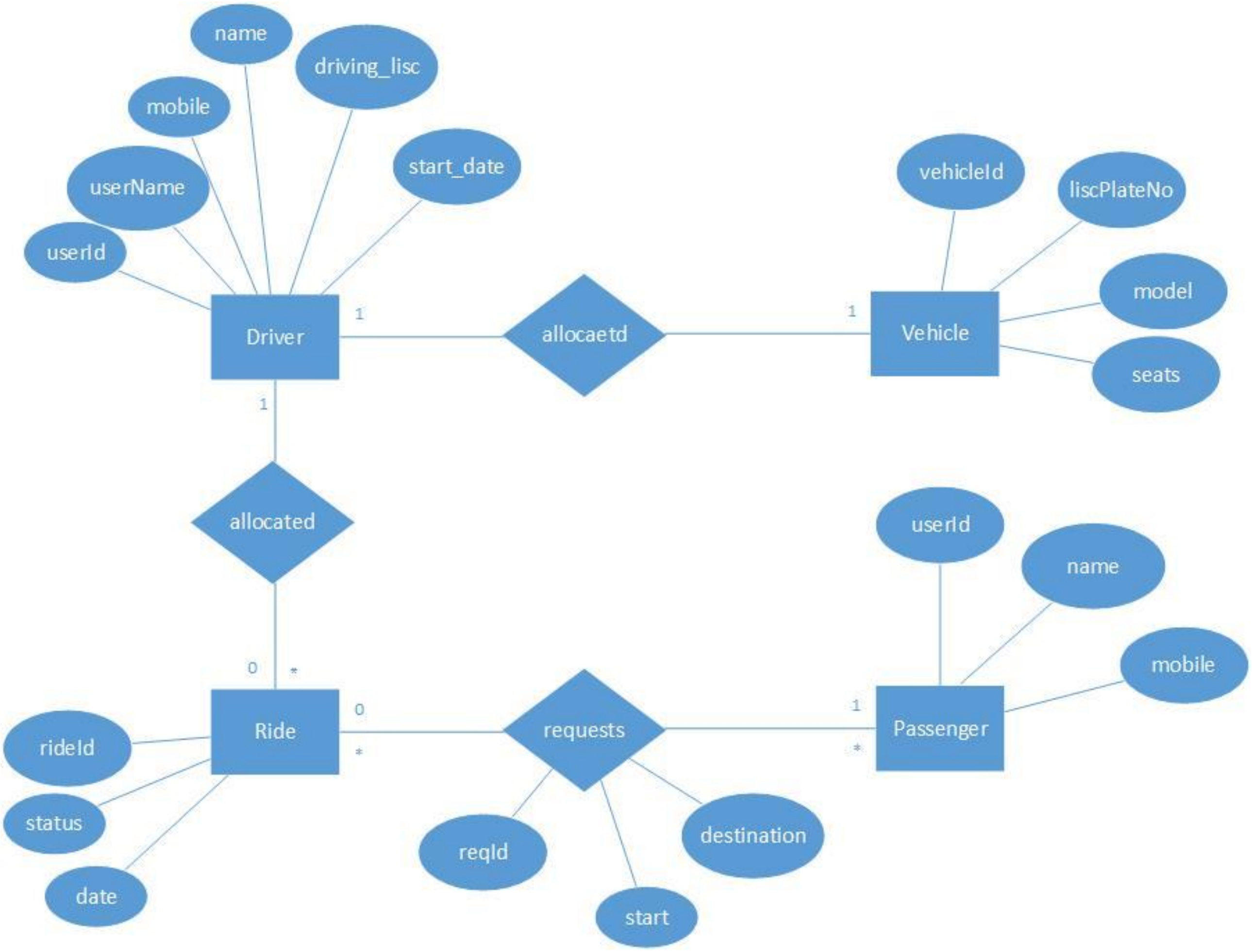


Figure 22: ER model for the ridesharing platform

The entity relational model can be used to identify the elements and fields the database should store, to start with.

```
//passenger
{
   userId: p01,
   name: "Lee Cooper",
   mobile: 2177777777
}
//driver
{
   userId: d01,
   userName: “hwolowitz”,
   name: “Howard Wolowitz”,
   mobile: 2177000000,
   driving_lisc: “D123456789”,
   start_date: 02/28/2019
}
//vehicle
{
   vehicleId: v01,
   liscPlateNo: “BB12345”,
   model: “Toyota Corolla”,
   seats: 4
}
//ride
{
   rideId: v01,
   liscPlateNo: “BB12345”,
   model: “Toyota Corolla”,
   seats: 4
}
```

In traditional relational modeling, a class diagram can be used to identify the skeleton of the database structure. To acquire a basic image of the database required, a class diagram was drawn following traditional object-oriented methodologies. Following figure 23 displays the class diagram for the database of the ridesharing application. It identifies the relationships between objects in this scenario, which are ride, request, driver, passenger and user. This illustration follows the standard class diagram notation.

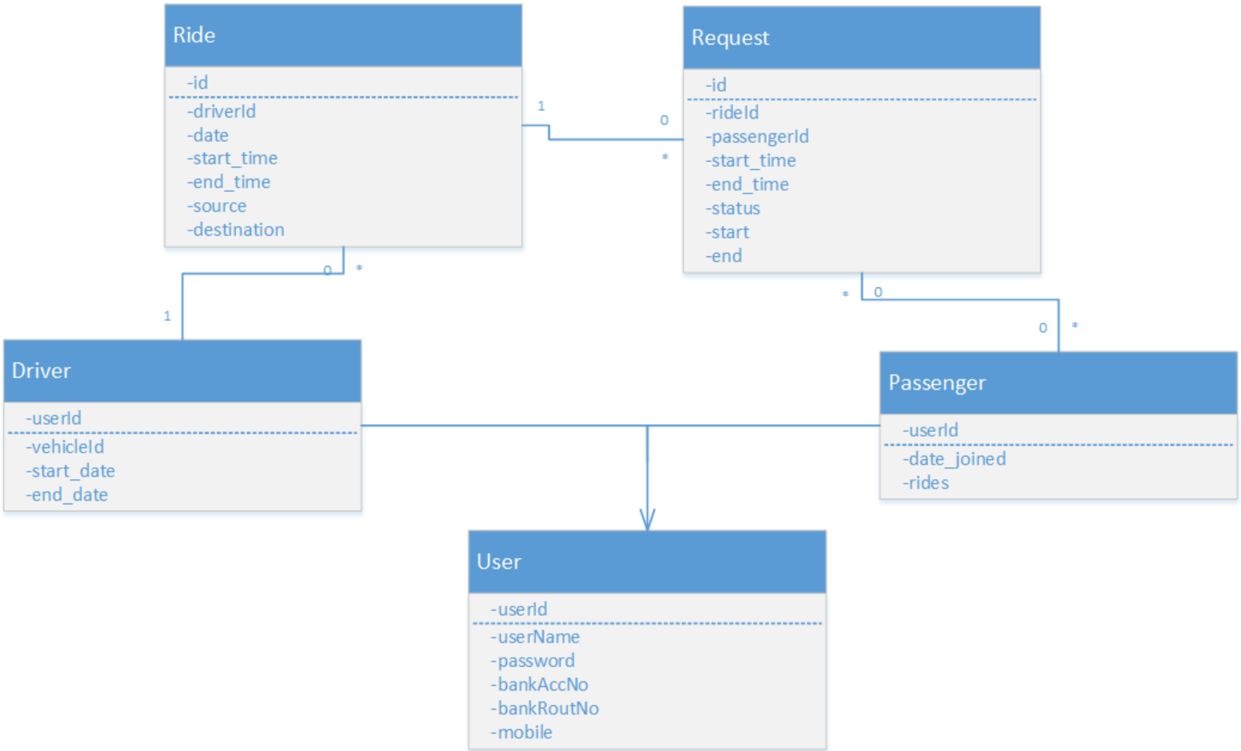


Figure 23: Class diagram for the Ride share application

To be successful, an application should be financially, practically and technologically feasible to the stakeholders. Therefore, some selected feasibility analyses were done to evaluate the usability of such an application and thus what features should be embedded with the database in the backend.

- Technical feasibility: the physical characteristics of the equipment and software used and the criteria for capturing, presenting and storing the digital surrogates.

  Response time

  Concurrent access

  Data communication with distant locations

- Operational feasibility: the application's usability in human and organizational aspects

  User friendliness

  Ease of implementation

  Ease of maintenance

Each available database application selected was studied evaluating its ability to match up the database requirements identified in this case study. Evaluating the application of these products included comparing several features which are present in the context of real-time data modeling. Following features were taken into consideration.

Location accuracy: The database should be capable of keeping track of the pick-up and drop-off locations of the passengers. Therefore, it is expected to store the coordinates of specific locations within the area of operation.

Reliability: The database should be able to ensure continuity of data delivery. Thus, it should preserve the temporal validity of data items that reflect the state of the entities which are being controlled by the system.

Adaptability: Compatible integration with other systems is a big benefit to a mobile based location technology platform. Mobile phones are equipped with different operating systems and different map applications attached to them. The real-time database systems are expected to work with existing hardware and software making it easier to parse together all data needed to create accurate information transfers. The database must adapt to certain changes in assignments without imposing more cost on the purchase of new hardware.

Cost: With the current market full of many open source mobile applications, cost reduction is a priority in each and every application. The database being expensive will pave the way for the system development to be expensive. Therefore, an open source real-time database is much suitable for this kind of an application.

Security: Because a ride share application comes with sensitive data like authentication details of its users, it is important to consider the risk of hacking and data loss before selecting the particular database.

Scalability: The installation of a real-time database infrastructure should avoid any disruptions in the use of the application, resulting in a temporary loss of passenger data or the need to deploy expensive hardware expansions when moving into other areas.

Hosting: The location of the data is to be considered while selecting a database because the hosting feature decides the transfer speed, security, and cost of the selected technology.

Update speed: The real-time database solution should be capable of capturing interactions between drivers and passengers within seconds to facilitate the accurate collection of ride information. The transactions should be processed to meet their deadlines accordingly.

Setup and configuration: The real-time database should be able to provide streamlined installation facility. It would be more advantageous and less time consuming if it requires minimum training. Also, it must be able to be managed and utilized according to needs of the users.

Technical support: Technical support is a crucial aspect necessary while choosing any technology. Users should be able to get customer care services at every point of time.

Concurrency: The database should be capable of handling the input from the environment simultaneously.

The following results and discussion section will evaluate those results.

6. Results and Discussion

To evaluate the performance of a real-time database in a ride share application, several features of selected databases were researched. Checking out these features will help us understand what type of database is most suitable for a ridesharing application.

Firebase real-time database was first evaluated for the database requirements for a ridesharing application.

Table 1 illustrates the features of Google Firebase, a typical key-value based real-time database. Firebase uses JSON (JavaScript Object Notation) tree as its data storage model. Google maps can be embedded in a Firebase database to define the location of a device. Firebase can be installed and used in almost all types of operating system environments, including mobile phone operating systems too. Security rules can be defined in Firebase using pre-defined libraries which support password, phone number authentication methods. The data in a Firebase database are stored in the Google cloud, thus supporting cloud hosting. The users can expand the number of servers and machine allocated, without affecting the database by sharding, which partitions data by key ranges and distributes the data among two or more database instances. Concurrent access for any type of application is supported by Firebase, up to a limit of 200K users. The operations on data can be performed even when the application is offline too. To install and setup a Firebase database for your application, a Google account is required, which will have 24x7 chat support for the users. Monthly charges for this product will apply after using free trial.

Feature	Observations
Data storage structure	JSON tree

Location accuracy	Google Maps features can be easily embedded
Adaptability	iOS, Android and Web supported
Security	Rules can be written according to the data and the application.
Hosting and data storage	Cloud hosting in Google cloud
Concurrency	200k concurrent user limit applies to each individual database
Scalability	Database sharding allowed.
Offline functionality	Supported
Setup and configuration	Google account required
Technical support	Pre-defined answers, 24h chat support and community support available
Cost	Storage is billed at $5 for each GB/month, evaluated daily

Table 1: Firebase features

Developers can use both Cloud Firestore and the Realtime Database together in the same project, which will result in a higher level of scaling. At the same time, up to 100 shards can be created, empowering the scalability of this product.

Thereafter, MongoDB real-time database was evaluated for the database requirements for a ridesharing application. Table 2 shows the considered features and observations for MongoDB. As JSON is a basic format of saving data, MongoDB has some extensions added to it; Canonical Mode and Relaxed Mode. The aim of adding these extensions is to preserve data type and emphasize readability and interoperability without losing information. Languages like C, C++, Java, Perl use the extended JSON format in MongoDB. Google maps can be embedded in a MongoDB database to define the location of a device. MongoDB can be installed and used in almost all types of operating system environments, including mobile phone operating systems. Security rules can be defined in MongoDB using pre-defined libraries which support encrypted authentication methods. The data in a MongoDB database are stored in the Google cloud storage, thus supporting cloud hosting. Other than that, a MongoDB database can be stored in the local disk storage too, supporting standalone applications. In a cloud hosting environment, the users can expand the number of servers and machine allocated, without affecting the database. In MongoDB, the system growth is facilitated by vertical and horizontal scaling, increasing the capacity of the server as required by the application architecture. Concurrent access for any type of application is supported by MongoDB, allowing multiple users to update the same data at the same time. The operations on data can be performed even when the application is offline. To install and setup a MongoDB database for your application, a MongoDB account is required, which will have 24x7 chat support for the users. Monthly charges for this product will apply.

Feature	Observations
Data storage structure	Extended JSON: extensions added to the basic JSON format
Location accuracy	Google Maps API can be used
Adaptability	iOS, Android and Web supported

Security	Pre-defined functions for Authentication, Authorization, TLS/SSL and Encryption are available.
Hosting and data storage	Cloud hosting in Google cloud, Storage in Cloud, device memory and on disk managed by multiple storage engines
Concurrency	Supported using locking and other concurrency control measures
Scalability	Database sharding to support deployments
Offline functionality	Supported
Setup and configuration	MongoDB account required
Technical support	Pre-defined answers, 24h chat support and community support available
Cost	Starting at $0.08/hr, estimated cost $56.94/month

Table 2: MongoDB features

Multiple authentication mechanisms are supported by MongoDB i.e.; SCRAM, x.509 Certificate Authentication, LDAP proxy authentication, Kerberos authentication. Direct authentication to a specific database shard is also supported. Role-Based Access Control (RBAC) to govern access to a MongoDB system is also facilitated.

Apache Cassandra was used as the columnar real-time database for the evaluation. Table 3 displays the features which were taken into consideration and the observations related. Cassandra supports Query-driven modeling, because joins are not supported. Each query run throughout the transactions is supported by a table, with duplication of data, paving the way to an efficient reading of data. But any built-in libraries or functionalities supporting map APIs could not be found in Cassandra development environment. Cassandra can be installed and used in almost all types of operating system environments, which support Apache server configurations. Security rules can be defined in Cassandra using pre-defined encrypted authentication methods. These features are disabled by default, unless enabled by the user. It needs studying and understanding the features before configuring the cluster according to the expected security concerns. The data in a Cassandra database are stored in RAM(Random Access Memory) and hard disks. Minimum production server requirement was of 8GB RAM. Concurrent access for any type of application is supported by Cassandra, handling simultaneous requests on several machines. The operations on data can be performed even when the server is offline. To install and setup a Cassandra database for your application, a pre-installation of Java Virtual Machine is required. It allocates the file size for the database tables. Cassandra also has 24x7 chat support for the users. This product comes free of cost.

Feature	Observations
Data storage structure	Query driven. CQL: Cassandra Query Language is a model somewhat like SQL in the sense that data is put in tables containing rows of columns. JSON documents are also supported in latest versions.
Location accuracy	No directly embedded plugins for maps APIs
Adaptability	iOS, Android and Web supported

Security	Pre-defined functions for TLS/SSL encryption for client and inter-node communication, client authentication, authorization are available.
Hosting and data storage	Device memory and on disk managed by multiple storage engines
Concurrency	Supported.
Scalability	Nodes may be added/removed as needed. Sharding requires additional application logic.
Offline functionality	Supported
Setup and configuration	Pre-installation of JDK/Java required
Technical support	Pre-defined answers, 24h chat support and community support available
Cost	Free

Table 3: Cassandra features

It is noted that Apache Cassandra is a query driven real-time database management system, therefore shows limited overall functionality comparing with Firebase and MongoDB.

To represent the graph model in the real-time database for the study, OrientDB was used and studied. Table 4 displays the features which were taken into consideration and the observations related to those. OrientDB supports disk, memory, local and remote storage, which can consist multiple clusters. But any built-in libraries or functionalities supporting map APIs could not be found in OrientDB development environment. OrientDB can be installed and used in almost all types of operating system environments, including the support of Java Virtual

Machine. A special role called ‘System User’ is used to implement security rules, apart from database encryption. Concurrency on OrientDB is rather complicated than the previous products. It does not allow simultaneous access to instances of databases. It uses Optimistic Concurrency control, which needs environments where conflicts between operations are rare. Each version of each record will be checked after each transaction for any updates from a device. The operations on data can be performed even when the server is offline. To install and setup an OrientDB database for your application, a pre-installation of Java version 8 or higher is required. Cassandra also has 24x7 chat support for the users. This product comes free of cost.

Feature	Observations
Data storage structure	Gremlin is the open source standard language for graph databases. NoSQL queries are used which are constructed using JSON objects.
Location accuracy	No directly embedded plugins for maps APIs
Adaptability	Compatible with any operating system that implements the Java Virtual machine like Linux, Microsoft Windows, from 95/NT and later, IBM AIX. Adaptability with Gremlin Server facilitates supporting different languages
Security	Database can be accessed by the server’s user. Database encryption ensures the security concerns

Hosting and data storage	External cloud services like Amazon Web Services, Microsoft Azure are supported. Local direct memory allocated by the JVM. Total size is limited by the -XX:MaxDirectMemorySize JVM option
Concurrency	Optimistic Concurrency Control(multiple transactions can compete frequently without interfering). But high concurrency on the same vertices is not well supported. It requires a reduction of transaction size.
Scalability	Hazelcast Open Source project is used for auto-discovery of nodes, storing the runtime cluster configuration and synchronizing certain operations between nodes. Sharding of data is done at class level, by using multiple clusters per class.
Offline functionality	Supported
Setup and configuration	Requires Java, version 8 or higher.
Technical support	Pre-defined answers, 24h chat support and community support available
Cost	Free

Table 4: OrientDB features

The most important functionalities for a ride share application were discussed in the previous case study. For our case, support with a map tool for tracking the users' locations is a primary requisite. Firebase and MongoDB most prominently support the Google map features. Since a ride share application is expected to be mobile friendly, cloud hosting of those two products will result in a more lightweight product. The four chosen products have different data models. But both Firebase and MongoDB are using JSON methodology, which is highly object oriented. Object oriented programming practices are easily adaptable for the developers. Therefore, it will be beneficial for the development environment to be adaptable to the object-oriented methodologies.

All the four products have addressed the basic security concerns of a real-time application. Firebase and OrientDB requires most of the security concerns to be queried and implemented, while the other products have many built-in security functions. Each of the products have sufficient concurrency control mechanisms for concurrency control.

Following the studies upon four major types of the real-time databases and with the requirement analysis done in the case study, a simple database was designed for the ride sharing application which was designed previously by Muhammad Rehaan Saeed, which has the most essential features. Setting up Firebase and MongoDB were easier than installing Cassandra and OrientDB, because of the smaller number of pre-requisites. Firebase real-time database in the Firebase console was studied and evaluated for the components, driver, passenger, and admin. Basic create, read, update, delete operations were coded and tested in the console. Notably, installing and using Google map API with Firebase was much easier than with other real-time databases analyzed for this study. MongoDB on the other hand was able to perform the same operations with a similar level of efficiency.

Apart from those two real-time databases, Cassandra and OrientDB were also taken into consideration, but with the limited resources raised the compatibility issues with the development environment, thus limiting the performance evaluation to two real-time database products.

7. Conclusion

With the evaluated products and results, a conclusion can be made that Firebase or MongoDB are well suited for a ride share platform. Even though they come with some ranges of payment plans, their features could not be ignored in comparison with the other products. The compatibility with Google maps, ease of understanding the technology behind the product and ease of querying for the ride share needs are most highlighted. Also, whenever concurrent transactions are flowing in, these products are built and designed to give an optimum performance. Choosing one of these products for an application would depend upon how the developer expects to host the application, and what type of storage he prefers. Since Firebase is a product offered by Google, it is proven to be highly reliable with a cloud-based application or project. On the other hand, if the application is expected to be stored in a local environment, MongoDB would be a better choice, because of its compatibility with the disk storage. The scalabilities of all four products are sufficient for a simple application like ride sharing. But for a more widened scope, a cloud based real-time database would be a better match.

8. Future Work

The major portion of this study was carried out based upon a literature survey and basic testing with simple prototypes. Testing each real-time database with applications built in Android and iOS mobile phone operating systems would pave the way to a better understanding in these concepts. The cost for building these sample projects was also a hurdle, because proper development keys to build Android and iOS applications have to be purchased. Also, access to a MAC personal computer is required to build an iOS application. These shortcomings should be addressed in carrying out further studies and tests.

Another hurdle faced was evaluating the payment options for a ride share application. Because installing and testing payment gateways come attached with cost, relevant evaluations could not be done. Therefore, a different mechanism is required to evaluate which real-time database would be a best match in the context of the payment option. Finally, requirements for mechanisms to provide security background checks on potential drivers, and provide insurance would be a good way to bring this study to a more professional level.

The ride share application developed by a graduate student in the School of Technology at the Eastern Illinois University was a prominent part of this study. Once all the features start working properly it is be available for the public to use. In the final phase of the ride share application, the in-app payment feature will also be incorporated for the hassle-free money transactions between the driver and the passengers.

A mechanism to evaluate the speed of transactions within the ride sharing scenario in each of these applications would have provided a much better conclusion for this study. Building up such a mechanism and do the performance testing will enhance the importance of the results of this study.

References

Buttazzo, G. C. (1997). *Hard real-time computing systems: predictable scheduling algorithms and applications*. Boston: Kluwer.

Brickell, A. (2019, June). Uber sees hypergrowth, vulnerabilities in its S-1. Retrieved from https://earlyinvesting.com/ubers-got-hypergrowth-growing-pains/.

Furuhata, M., Dessouky, M., Ordóñez, F., Brunet, M.-E., Wang, X., & Koenig, S. (2013). Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, *57*, 28–46. doi: 10.1016/j.trb.2013.08.012

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, *47*, 98–115. doi: 10.1016/j.is.2014.07.006

Heggie, J. (2019, January 8). Who benefits from big data? Retrieved from https://www.nationalgeographic.com/science/2018/12/partner-content-big-data-benefits/

Inc., G. (2014, September 22). Gartner's Big Data Definition Consists of Three Parts, Not to Be Confused with Three "V"s. Retrieved July 18, 2020, from https://www.forbes.com/sites/gartnergroup/2013/03/27/gartners-big-data-definition-consists-of-three-parts-not-to-be-confused-with-three-vs/

Ključnikov, A., Popesko, B., & Kloudová, J. (2019). Economics of the international ridesharing services - a trap for amateurs. *Entrepreneurship and Sustainability Issues*, *6*(3), 1172–1181. doi: 10.9770/jesi.2019.6.3(8)

Kumar, K. B. S., Srividya, & Mohanavalli, S. (2017). *A performance comparison of document oriented NoSQL databases*. 2017 International Conference on Computer, Communication and Signal Processing (ICCCSP).

Moroney, L. (2017). *The Definitive Guide to Firebase: Build Android Apps on Googles Mobile Platform*. Berkeley, CA: Apress.

Rouse, M., Vaughan, J., Rouse, M., Rouse, M., & Barney Beal. (n.d.). What is NoSQL (Not Only SQL database)? - Definition from WhatIs.com. Retrieved from https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL.

Shukla, P. (2016). NoSQL database: Cassandra is a better option to handle big data. *International Journal of Science and Research (IJSR)*, *5*(1), 24–26. doi: 10.21275/v5i1.nov152557

Strauch C.(n.d.). NoSQL Databases. Retrieved from https://www.bookcord.com/Data/Science/Computer/NoSQL-Database_(bookcord.com).pdf

Wang, Y., Wang, S., Wang, J., Wei, J., & Wang, C. (2018). An empirical study of consumers' intention to use ride-sharing services: using an extended technology acceptance model. Transportation. doi: 10.1007/s11116-018-9893-4

Xing, X., Warden, T., Nicolai, T., & Herzog, O. (2009). SMIZE: A Spontaneous Ride-Sharing System for Individual Urban Transit. *Multiagent System Technologies Lecture Notes in Computer Science*, 165–176. doi: 10.1007/978-3-642-04143-3_15

Amazon Kinesis Data Streams (n.d.). Retrieved from https://aws.amazon.com/kinesis/data-streams/

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

Firebase. The Real-time App Platform (n.d.). Retrieved from https://stackshare.io/firebase

JSON File Extension (n.d.). Retrieved from https://fileinfo.com/extension/json

MIT.(n.d.). Superheroic JavaScript MVW framework. Retrieved from https://angularjs.org/.

MIT.(n.d.). Backbone.js. Retrieved from https://backbonejs.org/#Getting-started.

MongoDB Inc.(n.d.). The database for modern applications. Retrieved from https://www.mongodb.com/

NoSQL databases vs graph database comparisons: Neo4j. (n.d.). Retrieved from https://neo4j.com/developer/graph-db-vs-nosql/.

Oracle Coop.(n.d.). What is big data?. Retrieved from https://www.oracle.com/big-data/guide/what-is-big-data.html

Uber Revenue and Usage Statistics (2020). (2020, March 24). Retrieved from https://www.businessofapps.com/data/uber-statistics/

UpGuard. (2019, July 30). The cost of downtime at the world's biggest online retailer. Retrieved from https://www.upguard.com/blog/the-cost-of-downtime-at-the-worlds-biggest-online-retailer. 2019 Database trends – SQL vs. NoSQL, top databases, single vs. multiple database use. (n.d.). Retrieved from http://highscalability.com/blog/2019/3/6/2019-database-trends-sql-vs-nosql-top-databases-single-vs-mu.html.