Migrating From Traditional To Document-Oriented Databases: A Literature Study

Ishank Jain
Department of Computer Science
University of Waterloo
Waterloo, Canada
ishank.jain@uwaterloo.ca

***Abstract*—There has been an exponential growth in the generation of data over the years, the traditional databases are no more able to be as productive as they were use to be earlier. Moreover, the advent of large complex datasets, NOSQL databases have gained popularity for their ability to efficiently handle such datasets in comparison to relational databases. There are a number of Documented-oriented databases (also referred as data stores) for e.g. MongoDB, Couchbase, ArrangoDB etc. Operations in these data stores are executed quickly. Systems that provide the efficiency of a schema-less interface are widely used by users in mobile and cloud applications. This paper describes the background, characteristics, data model of NoSQL databases. Further, the conventional Documented-oriented databases are separately described in detail, and extract some properties to help to choose Documented-oriented databases. The aim of this paper is to provide the readers with an extensive background for understanding the benefits of migrating data from traditional to document-oriented databases [6].**

***Index Terms*—ArrangoDB, Non-relational databases, relational databases , NoSQL, MongoDB, Sharding, COucheDB, consistency, optimization.**

I. Introduction

The data overload, originating from the growing volume of Big Data during the past decades, requires the introduction and integration of new processing approaches into everyday systems and activities (ubiquitous and pervasive computing) (Cook Das, 2012; Kwon Sim, 2013) [9]. Processing large volume of data manually is very restrictive.

NoSQL includes various types of databases which includes key-value databases, column-oriented databases, graph-based databases, and document-oriented databases. The need for the NoSQL arises from the incompatibility of relational databases to handle large volume of data over distributed network and to cope with new trending technologies like cloud computing, big data, distributed processing etc. [11].

A. Advantages of NoSQL Databases

The various advantages of NoSQL databases in comparison to traditional databases are listed as follows:

1) Horizontally scalability: NoSQL databases are able to distribute the data and processing load evenly to multiple servers and helps to improve the performance by horizontally scaling up the data.
2) Schema-free design:In document-oriented databses, there is no need for prior establishment for storing data fields and no need to define any fixed structure to the data sets as was in the case of relational databases.
3) Low cost : In NoSQL databases, the cluster can be expanded easily by adding new node, NoSQL databases run on server cluster whose expansion is generally cheap.
4) Integrated Caching Facility: NoSQL databases are able to cache data in system memory to raise their performance and increase data output. [8]

Document-oriented databases are generally aimed for managing, retrieving, and storing document and semi-structured data in industries, where the amount of data is very large and distributed. Document-oriented databases are one of the main categories of NoSQL databases. In general, they all assume documents encode and encapsulate information (or data) in some standard format or encoding. Encodings (file formats) can include XML, YAML, JSON and BSON depending on the system. Partitioning the databases appropriately is an important step that determines the efficiency of sharding (distributing data on the server). This involves choosing an index for the document, competently as a shared key is needed for further horizontal scaling of the database.

NoSQL (also called, Non-relational) databases come mainly in four different types:

1) Key-Value stores: The concept here is to use a hash table and providing each item of data with a unique key and a pointer; data can only be queried using key. This model is useful in representing polymorphic and unstructured data but is inefficient when the application is only interested in querying or updating part of a value.
2) Column-oriented databases: The database model was developed particularly to process huge amounts of data that is spread across multiple servers. They store data using a sparse, distributed, multidimensional sorted map.
3) Graph stores:This database model is modeled as a network of relationships between specific elements of graph structures like nodes and edges to represent data.
4) Document-oriented databases: document oriented

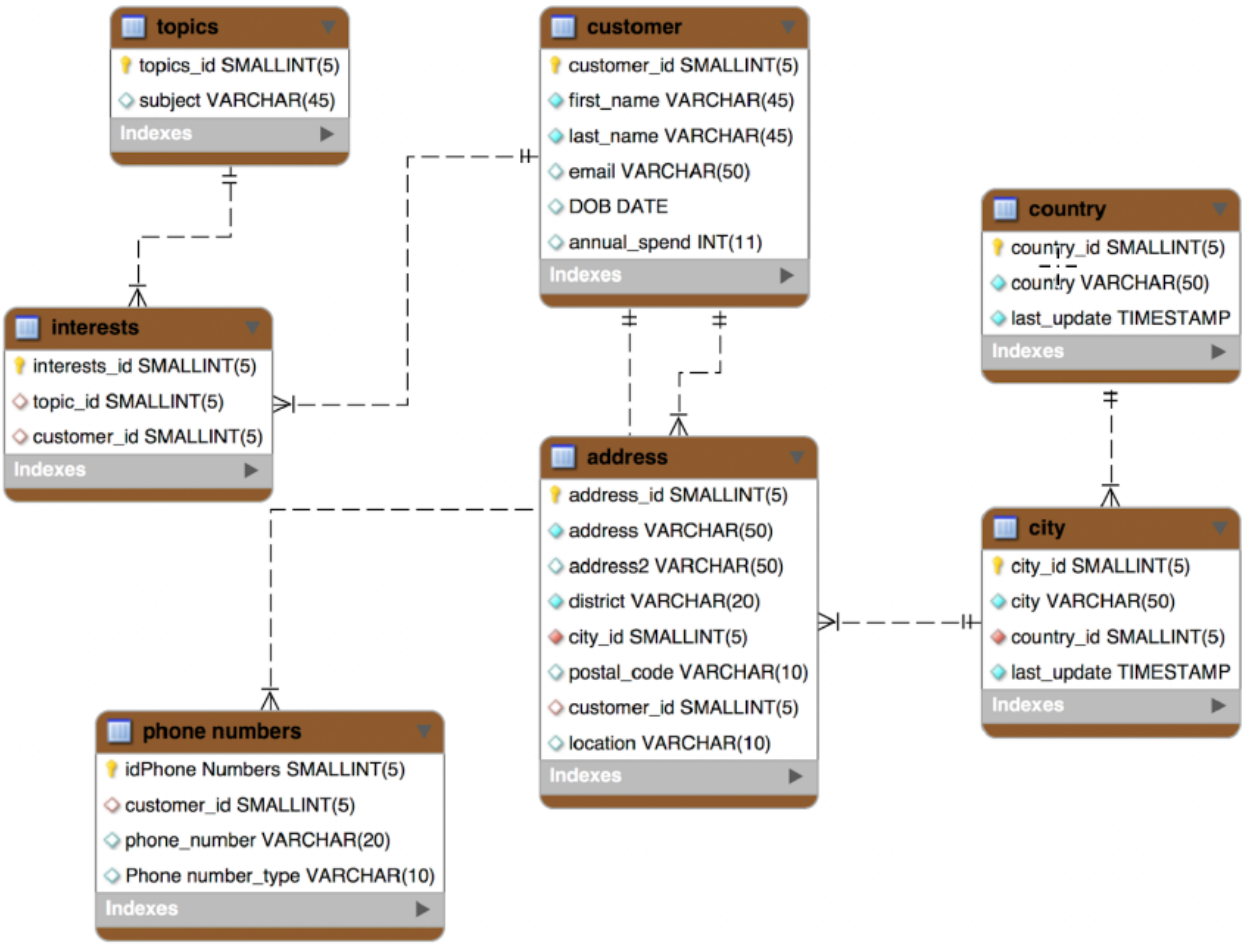

Fig. 1. Relational Database Model

databases store data in documents format. These documents typically use a structure similar to JSON(JavaScript Object Notation); they provide a natural way to model data that is closely aligned with object oriented programming.

Before the user can add any data records to a SQL database, one must first define the types of the values that make up a record. A document store does not require an upfront specification of the types of documents that will be stored in it. A document that is added to a store can consist of one or more attributes and, with the exception of a unique identifier, no two documents need to have the same number and types of attributes.

SQL provides simple functions such as counting and aggregation, while a SQL database might provide a means for adding custom processing code that does some complex computation on query results. The expense of handling query results can increase rapidly as the database develops and the calculation's complexity rise. To moderate this cost, calculated values can be cached for situations where the values are required as often as possible .

When the factors such as fast look-ups, scalability, availability, replication, easy maintenance, etc are considered, using these factors through Relational Database Systems -it seems to get tougher. There is a range of document-oriented databases, for instance, MongoDB, ArrangoDB, Couchbase, DynamoDB etc..In this paper, we discuss the concept of NoSQL databases and further we discuss the category of document-oriented databases on the basis of design, scaling techniques, availability and optimization techniques.

This paper is organized as follows: Section II presents the Motivation and Rationale of the study. Section III Research questions. Section IV Reseach methodolgy, Section V Benefits of data to document database. Lastly, section VI provides results and our findings. Then, Section VII presents the conclusion, followed by future work.

II. Motivation and Rationale of the study

I got the inspiration for this studies for the reason that proof/study is required for systems to gather acknowledgment in the industry. This survey is important to uncover the advantages of Document-stores and reveal usefulness they offer over conventional databases. Thus, there was a need to supply the readers with a outline on document stores. We expect that this paper fills that hole [3].

The value of any study can be understood from how well it elaborates on previous work, also from the studys intrinsic properties. Thus, combining all the unbiased, credible results from previous research would be a step towards understanding and expanding our knowledge on the domain. Similarly, the rationale of our study was to present credible results and benefits from the overwhelming amount of publications through a critical exploration and evaluation of the previous studies and research.

To bridge the gap from what we know about databases like MySQL, to what we may not yet understand about document databases. While a SQL database is made up of one or more tables and each table is made up of one or more columns, a NoSQL document store is essentially a single container. Every document that is added to the store is added to this one container.

Data is at the heart of every application. Therefore, there is a need to understand the functionalities, data types, and system specific optimizations that document-oriented databases offer.

III. Research Questions

The goal of this paper is to develop a literature survey on data migration benefits from relation database into a document-oriented database. we will addressing following research question in our paper:

1) What are the problems that user face while migrating from relation database to a document-oriented database?
2) What are the gaps that needs to be addressed in traditional relational databases such as file types?
3) What are the data modeling possibilities, including options that are available?
4) what are the query options, scaling options, fault tolerance mechanism, data dumping, and system specific optimizations?
5) How to compare the popular and most implemented databases such as MongoDB, Couchbase, ArrangoDB?

IV. Research Methodology

We conducted the literature study in which consists of five different stages:

- looking for the literature collecting data,
- assessing the search results selection of academic papers for primary studies,
- Reviewing the papers,
- Synthesizing the results
- reporting the results.

In the first stage, my goal was to gather the appropriate academic papers for studies. For that reason, I declared the key search terminology. I extensively and iteratively searched popular databases for academic resources, this includes VLDB - International Conference on Very Large Data Bases, IEEE - the Institute of Electrical and Electronics Engineers, IJCA - International Journal of Computer Applications, CEUR Workshop Proceedings, and ACM Digital Library. The search terms included Databes, NoSQL systems, Document-stores literature review, Document-oriented systems, JSON file system etc.. The time frame of the search for the review is bound within the last seven years (2012-2018), in which emergence and adoption of Documnent-oriented database has grown rapidly [3].

The scope of the paper is limited to the papers included in research between time period 2012-2018 due to time constraints. The scope is further limited to the three systems (MongoDB, COuchbase, ArrangoDB) included in the research based on their popularity and implementation. I decided not to include PostgreSQL in the report as initially planned because of its limited popularity and implementation as a document-oriented database.

V. Benefits of migrating to Document databases

Data accumulated by data systems is one of the important assets for any company. Pushed by customer demands and pulled by changes in technologies, companies from time to time migrate from one data system to another.

There are several benefits of using document-oriented database over relational database systems:

1) Support for unstructured text: Being able to manage unstructured text greatly enhances information and can help organizations make better decisions. For example, advanced uses include support for multiple languages with facetted search, snippet functionality, and word lemmatizing and stemming support.
2) Ability to handle change over time: If a document structure changes, dynamic schemas in NoSQL databases removes the need of schemas to start working with data.
3) No reliance on SQL: Application users dont need to know the inner workings and vagaries of databases before using them. NoSQL databases empower users to work on what is required in the applications [13].
4) Ability to scale horizontally: NoSQL databases handle partitioning (sharding) of a database across several servers. So, if users' data storage requirements grow too much, user can continue to add inexpensive servers and connect them to database cluster (horizontal scaling) making them work as a single data service.
5) Support for multiple data structures: Many applications need simple object storage, whereas others require highly complex and interrelated structure storage. NoSQL databases provide support for a range of data structures.
    - Simple binary values, lists, maps, and strings can be handled at high speed in document databases.
    - Highly complex parent-child hierarchal structures can be managed within document databases.

The points mentioned above are some of the key benefits of using document databases over traditional relational databases.

VI. Results

Relational, tabular databases have had a long-standing position in most organizations. This have resulted in a default way to think about using, storing, and processing enriching data. But companies are increasingly encounter limitations of this technology.

Organizations need a fresh way to work with data. In order to handle the complex data of modern applications and simultaneously increase development velocity, the key is a platform that is:

1) Easy, let us work with data in a natural, intuitive way, supported by strong data consistency and transactional data guarantee,
2) Flexible, so that they can adapt and make changes quickly
3) Fast, delivering great performance with less code
4) Versatile, supporting a wide variety of data models, relationships, and queries

A. Database concepts

1) Relational Databases: The data model in a traditional relational database management system (RDBMS) utilizes the controlling metaphor in form of a table, Records are stored in tables, with each record being a row in that table. The column heads define the available fields and the values for each of the columns per record in individual cells. Such tables can be organized in databases (e.g., for multitenancy use). Data in a relational database system can be queried and retrieved in a tabular format. It can be full rows or subsets of columns.

Schemas can be modified in most RDBMS after the initial definition, but in some systems, especially with clusters, it can be an expensive operation because the data needs to be re-organized on schema changes. Data queries executed on a huge dataset, without using indexes can be significantly slower. Therefore, users are dependent on the database users for a well-defined and proper indexes for the best performance based on the data content and how it will be used.

2) Document Databases: JSON objects (and similar object formats) in a document store supports as documents (refer fig.**??**). JSON file supports all traditional and many compound data types that can be used in combination to form a complex data structures. Each document is essentially a JSON object at the surface level, but with arbitrary named attributes that can have traditional values or be nested arrays and objects. The document database does not require user to declare or define field attributes like SQL. Nor is it necessary to specify in advance the data types of fields [13].

Documents can also have an user-defined structure and can use any data type supported by the database. As a general idea, each document in a collection (group of JSON data objects) can be structured differently. However, there are usually some fields that all documents have in common in the sense of attribute keys. The attribute values may use different data types, nonetheless, even if the attribute keys match. Each document requires a key that uniquely identifies it within a collection. It can be assigned by the user upon creation, or can be created by the system. Document keys and document identifiers are always indexed. The index on the key attribute is called a primary index. It exists for each collection and cant be removed. Its a hash index with the options non-sparse and unique. This means it is always non-empty and that there are no duplicate keys in the collection [10].

There are several indexes type that are used in various document-oriented databases: Hash, Skiplist, geospatial, text etc.

B. Data modeling

Building upon the ease-of-use, flexibility, and speed of the document model, enables the users to satisfy a range of application requirements, both in the way data is modeled and how it is queried.

The flexibility and various data types in a document-oriented database makes it possible to model data in many different structures that is representative of the real world entities such as a music libraries. The document stores allow embedding of arrays and sub-documents that makes documents very powerful at modeling complex relationships and hierarchical data, with the ability to manipulate deeply nested data without the need to rewrite the entire document, this also helps to keep remodelling cost down. But documents can also do much more: they can be used to model table structures, key-value pairs, text, geospatial data, the edges & nodes used in graph models i.e. it provides freedom to choose the structure of the document rather then having a fixed schema [12].

1) Relational Databases: The most important step in building a logical model is to determine the entities which will be served by the database. After that the next step is to define the properties of these entities and their relationships i.e. hierarchy. An entity-relationship-model (ERM) is the standard way of representing all the relationships among the table in SQL databases. Relationships between records can be expressed with matching values between tables. They can be one-to-one relations (i.e, 1:1), one-to-many relations (i.e., 1:m) etc. To make use of stored relations, to resolve foreign keys to fields from another table, the tables or rather subsets of their records are joined on the fields which hold the record keys.

2) Document Databases: A semi-structured and unstructured data format can store data in XML, JSON, YAML, and BSON data formats. Another key point document database can store semi-structured data as a key-value without constraints; therefore, the value of the key can be any data type that gives flexibility to the database to store any type of data [5]. A semi-structured data is organized without the need for structured data formatting, which associates data models with relational or other data forms. JSON document model and the data structures can be used in object-oriented programming, which can further speed up user productivity [4].

To begin it is necessary to explain how a JSON objects look like. A JSON object has Curly brackets that surround JSON

Fig. 2. JSON Data Model

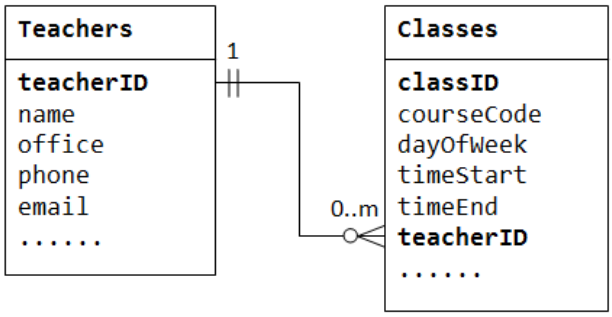

Fig. 3. RDBMS relation

objects, which can contain attributes. Attributes are given in pairs, key/value pairs. Each key and value is separated by a colon. Attribute pairs are separated by commas (,). Attribute keys are enclosed in double quote marks, which signifies a string. Keys are always string data types in JSON. The values can be of any type. Square brackets surround value lists. Refer Fig.4.

Document store allows multiple levels of nesting, which can be very powerful, but its not the only way to structure data. Each type of entity can be stored in a collection (group of JSON objects), Entities or documents can then be linked in multiple ways. Instead of establishing a connection with a pair of foreign keys, the from and to attributes of an edge can be used to express the relation and open the possibility for graph traversals.

3) Comparison: Data is modeled as tables for an RDBM system. Its common practice to normalize an initial model so that one arrives at a model without redundancies with atomic values, in the sense that pieces of information arent being split into separate tables and columns. On the other hand, Document database allow for simple and complex alike, independent for each document. Everything expressible with tables can be translated to JSON documents, without the need to specify in advance the structure of documents.

A direct conversion of relational records to documents would result in JSON documents with top-level attributes only (i.e., no nested objects). The possibility of having sub-attributes and arrays, however, enables quite a different data modeling approach. This approach permits the embedding of data in documents and eliminates some extra tables or collections.

```
{id: 1,
 name: Joe,
 url: '...',
 stream:
 [{
    user: 2,
    title: 'today',
    body: 'go fly a kite',
    likes: [
     {user: 3},
     {user: 1},
    ],
 }]
}
```

Fig. 4. JSON format

A JSON document containing multiple entities embedded in a single document could look like the example above (refer fig.4).

C. Query Concepts

Storing data is only part of the function of a database system. The ability to retrieve data in various ways from a database is where an document database is most useful. To facilitate the retrieval of data, a query language is provided. Relational database systems like MySQL use a Structured Query Language (i.e., SQL), while document databases provide system specific modification of JSON query language (JAQL) (such as AQL in ArrangoDB), or XQL (XML Query Language) and more.

1) Relational Databases: SQL can retrieve records from the RDBM system, it can also delete, update and insert records. It also allows user to create Tables and databases, which can be altered and dropped, and user permissions can be managed with SQL statements. These uses are grouped and classified based on their similarities: data retrieval queries as DQL (data query language); inserting, updating and deleting data as DML (data manipulation language); structure and schema statement as DDL (data definition language); and creating users and setting permissions as DCL (data control language). There also exists stored procedures for more complex logic [1].

Most SQL databases implement a subset of these standards operations or a slightly different variant with additional system-specific features in a query language. SQL system comes with several language constructs to interact with databases. The SQL statement can be read like an English sentence with a fixed order of clauses. The clauses must appear in the specific place if it is used. For instance, it would reusult in a syntax error if the FROM clause was coded before the SELECT clause.

The most commonly used join in SQL is the inner join, which will returns the intersection of two sets. An attribute column of the first table is used to match the values of a table with a specified column of another table. For instance, if there's a record with a specific value in a particular column in one table, and a record with the same value in a corresponding column in another table, those records with the selected columns are returned.

2) Document Databases: Most of the NoSQL databases allow RESTful interfaces for the data, many others offer query APIs. Since most systems have there own specific query tool, in this section, I have discussed AQL query language for ArrangoDB.

ArrangoDB has its own query language called AQL. It is a language that allows for looping through data, which is typically available in programming languages more than database query languages. Users can do both query and program in one flexible language environment. The most common form is the FOR loop used with a collection of documents, similar to SELECT in SQL (refer fig.5). It can be used to iterate through arrays and list, such as a list of attributes, for intermediate results from a sub-query or a variable defined inline. AQL provides which allows different nested structure using a FOR loop. Joins can be implemented by processing two collections (group of JSON objects), usually combined with a condition such as the equality of an attribute between both collections.

Other essential high-level operations are LIMIT, SORT, and FILTER. They can be coded in different locations, usually with varying results based on the processing order described by the query. The user can define what a query should return in a RETURN statement. One can chose which parts of a document to return, or the document as a whole, optionally with additional or altered attributes computed. Data modification queries (INSERT , DELETE , REPLACE , UPDATE) dont require a RETURN statement. Its mandatory only in data access queries.

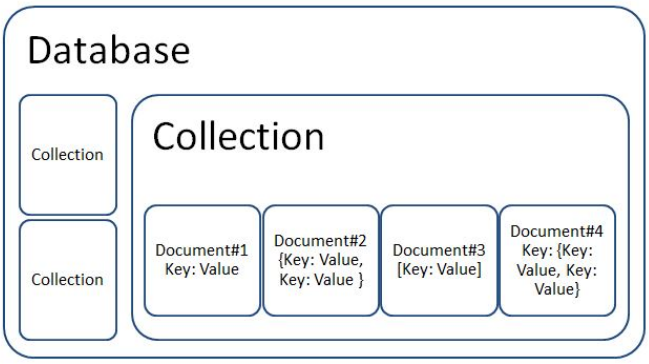

Fig. 5. Collection in a document database

It can be used nonetheless in modification queries, such as for returning old or new state of documents.

```
FOR s IN Song FILTER s.Title == "Tribute"
  // We want to find a Song called Tribute
  FOR album IN 1 INBOUND s PartOf
    // Now we have the Album this Song is released on
    FOR genre IN 1 OUTBOUND album Has
      // Now we have the genre of this Album
      FOR otherAlbum IN 1 INBOUND genre Has
        // All other Albums with this genre
        FILTER otherAlbum.year == album.year
          // Only keep those where the year is identical
          RETURN otherAlbum
```

Fig. 6. Query structure in ArrangoDB

In order to execute the query described in Fig.6, ArangoDB uses a two-phase query optimization technique. In the initial step, the query is parsed and transformed into an internal representation, an abstract syntax tree (AST). Then optimizer rules are used to transform the AST without changing the result. For instance, the optimizer will try to use indexes wherever possible. There are often several possible ways to execute the same query. For example, in a join it could start the search in any of the involved collections. The optimizer estimates whats required to solve the query for each way possible, and will execute the one that requires the lowest amount of operations.

3) Comparison: SQL syntax has a fixed structure, whereas high level operations in AQL, such as filters, can be put in various places for different purposes and results. The flow is more logical than adhering to SQL statement order. Plus, high-level operations like FILTER and SORT are more versatile. FILTER equals WHERE and HAVING in SQL, depending on where it appears. SORT is a direct equivalent of ORDER BY in SQL. LIMIT is the same as in MySQL, although there are some things to consider regarding scope in conjunction with subqueries. RETURN doesnt exist in SQL, but user can specify which fields to return in SELECT . In SQL what is to

be returned is given early in the statement, whereas in AQL, RETURN is placed at the end of a query and of subqueries.

Operations such as Aggregation with COLLECT is similar to GROUP BY statement in SQL, but the existence of compound types in AQL makes it very different. FOR loops in all the different variants is a key concept of AQL. Its an important piece and adds a sense of flow to queries. Joins, which are heavily used in most applications based on an RDBMS, are also possible in AQL. Joins are expressed with nested FOR loops combined with FILTER , instead of a separate syntax for each JOIN. Using nested FOR loops in AQL allows for the traversing of multiple levels of attributes within a document, while recursion is not possible in SQL [1].

D. Scaling-Out and Data Locality

Vertical and horizontal scalability is a essential topic and an increasing important one. Trying to accommodate increasing data volumes and number of user with a database running on a single server means users can rapidly hit a scalability wall, necessitating significant application redesign and custom coding work. Thus, horizontal scale-out for databases on low-cost, distributed network is performed using a technique called sharding. Sharding automatically partitions and distributes data across multiple physical instances called shards. Each shard is backed by a replica set to provide always-on availability and workload isolation. [7]

Unlike traditional relational databases, sharding is performed automatically and is generally built into the database system. A User don't face the complexity of building sharding logic into the application code. By simply hashing a primary key value, many distributed databases randomly distribute data across a cluster of nodes (distributed servers), imposing performance penalties (block time) when data is queried, or when data needs to be make available to specific nodes it adds to application complexity .

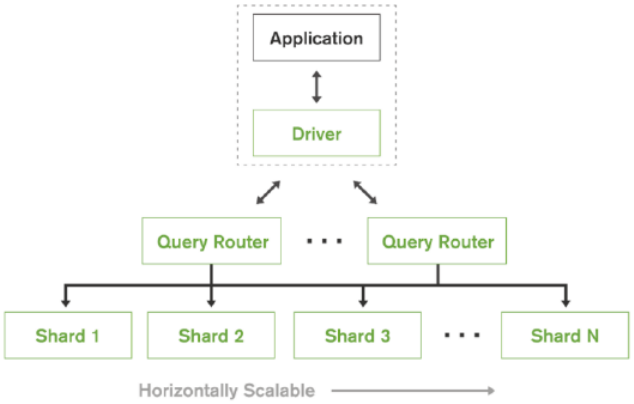

Fig. 7. Scale-out model ArrangoDB

To understand the sharding option that are available in a document database, in this section, I Will discuss sharding techniques which are specifically available in MongoDB [7];

- Ranged Sharding: Documents are partitioned across shards according to the shard key value. Documents with shard key values close to one another are likely to be co-located on the same shard. This approach is well suited for applications that need to optimize range based queries, such as co-locating data for all customers in a specific region on a specific shard.
- Hashed Sharding: Documents are distributed according to an MD5 hash of the shard key value. This approach guarantees a uniform distribution of writes across shards, which is often optimal for ingesting streams of time-series and event data.
- Zoned Sharding: It provides the ability for the user to define specific rules governing data placement in a sharded cluster. Zones are discussed in more detail in the following Data Locality section of the guide.

Other databases provide a few different sharding techniques as suited for there system.

E. Data Locality and Availability

As the user engagement increases online via web and mobile applications, availability becomes a major, if not primary, concern. Sharding allows precise control over where data is physically stored in a cluster. This allows the users to accommodate a range of application needs for example controlling data placement by geographic region for network latency and data co-ordination needs, or by hardware configuration and application feature to meet a specific class of service. Data Data placement rules can be continuously refined by modifying shard key ranges.

In comparison to a RDBMS, a distributed, document database partitions and distributes data to multiple database instances to online servers with no shared resources which improves the availability and processing capabilities. Moreover, the data can be replicated (duplicate data) to one or more instances for high availability (inter cluster replication). It is tough to ensure global availability for RDBMS where separate add-ons are necessary; which increases complexity, or failover because only a single data center is active at a time.

A distributed, NoSQL database includes built-in replication between data centers no separate software is required. In addition, some include both unidirectional and bidirectional replication (refer fig.8, fig.9) enabling full active-active deployments to multiple data centers; enabling the database to be deployed in multiple countries and/or regions and to provide local data access to local applications and their users. This not only improves performance, it enables immediate failover via hardware routers applications dont have to wait for the database to discover the failure and perform its own failover [2].

VII. System Specific Optimizations

In this section I will be discussing specific optimizations that are available in the three document databases discussed mentioned earlier in this paper.

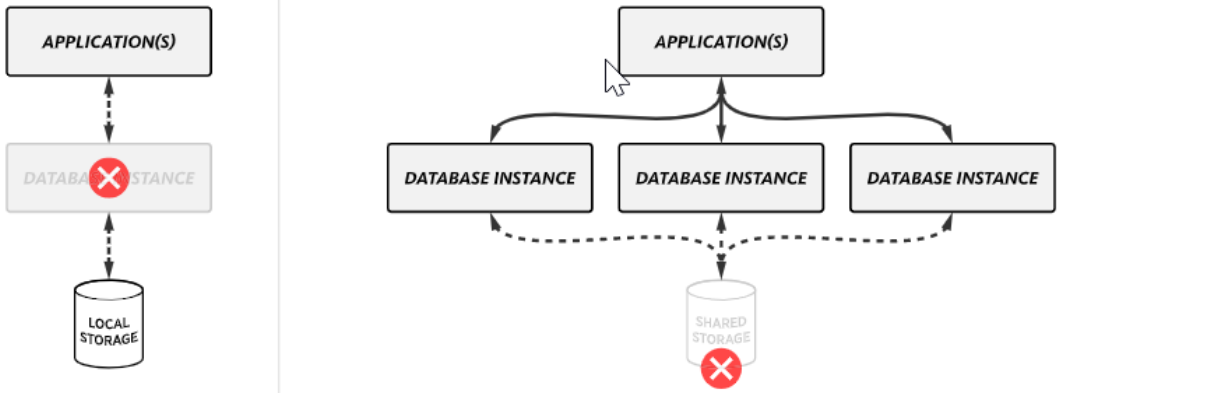

Fig. 8. Couchbase data availabilitys

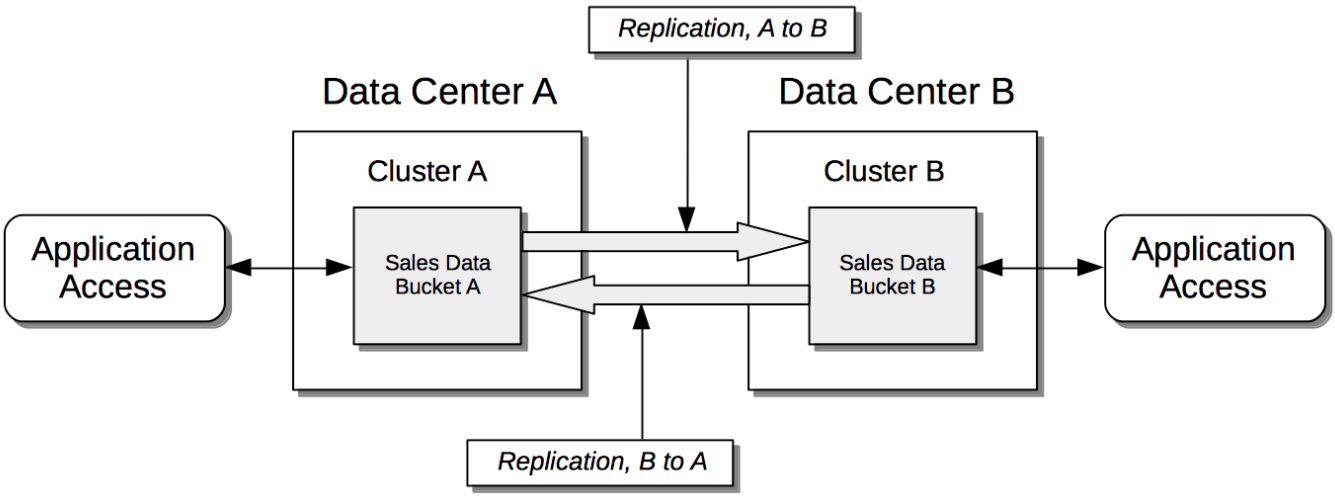

Fig. 9. Couchbase: Replication be-tween data centers

A. *MongoDB*

MongoDB stores data as JSON (JavaScript Object Notation) documents in a binary representation called BSON (Binary JSON). Unlike other databases that store JSON data as primitive strings and numbers, the BSON encoding extends the JSON representation to include additional types such as int, long, date, floating point, and decimal128 the latter is especially important for high precision, lossless financial and scientific calculations. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data, and sub-documents [7].

MongoDB provides native drivers for all popular programming languages and frameworks to make development easy and natural. Supported drivers include Java, Javascript, C/.NET, Python, Perl, PHP, Scala and others. The fig.10 shows the code efficiency between SQL an MongoDB code.

MongoDB offers a rich query fuctionality, it includes query types like Expressive Queries, Find, Check list, Geospatial, Text Search, Faceted Navigation, Aggregation, compute min, max, and average, Native Binary JSON Support etc. Similarly it also supports various indexing techniques such as Primary Index, compound index, geospatial index, text index, multi-key index etc. MongoDB supports variety of sharding techniques as discussed in section VI results under subsection Scaling-Out and Data Locality.

Having the freedom to put data where its needed enables users to build powerful new classes of application. However, they must also be confident that their data is secure, wherever it is stored. For this purpose MongoDB provides extensive capabilities to defend, detect, and control access to data:

SQL

```
def addUser(connection, user):
    cursor = connection.cursor()

    customerInsert = (
        "INSERT INTO customer (first_name, last_name,
            email, "
        "DOB, annual_spend) VALUES "
        "(%(first)s, %(last)s, %(email)s, %(dob)s, %(spend)
            s)")

    customerData = {
        'first': user['name']['first'],
        'last': user['name']['second'],
        'email': user['email'],
        'dob': user['dob'],
        'spend': user['annualSpend']
    }

    cursor.execute(customerInsert, customerData)
    customerId = cursor.lastrowid

    cityQuery = ("SELECT city_id FROM city WHERE city = %(
        city)s")
    for address in user['address']:
        cursor.execute(cityQuery, {'city': address['city']}
            )
        city_id = cursor.fetchone()[0]

        addressInsert = (
            "INSERT INTO address (address, address2,
                district, "
            "city_id, postal_code, customer_id, location) "
            "VALUES (%(add)s, %(add2)s, %(dist)s, %(city)s
                , %(post)s, %(cust)s, %(loc)s)")

        addressData = {
            'add': address['number'],
            'add2': address['street'],
            'dist': address['state'],
            'city': city_id,
            'post': address['postalCode'],
            'cust': customerId,
            'loc':  address['location']
        }
        cursor.execute(addressInsert, addressData)

    picQuery = ("SELECT topics_id FROM topics WHERE
      subject = %(subj)s")
    terestInsert = (
      "INSERT into interests (topic_id, customer_id) "
      "VALUES (%(topic)s, %(cust)s)")

    r interest in user['interests']:
      topicId = 0
      topicData = {
          'subj': interest['interest']
      }

      cursor.execute(topicQuery, topicData)
      row = cursor.fetchone()
      if row is None:
          topicInsert = ("INSERT INTO topics (subject)
              VALUES (%(subj)s)")
          cursor.execute(topicInsert, topicData)
          topicId = cursor.lastrowid
      else:
          topicId = row[0]

      interestData = {
          'topic': topicId,
          'cust': customerId
      }
      cursor.execute(interestInsert, interestData)

    oneInsert = (
      "INSERT INTO `phone numbers` (customer_id,
          phone_number, `Phone number_type`) "
      "VALUES (%(cust)s, %(num)s, %(type)s)")
    r phoneNumber in user['phone']:
      phoneData = {
          'cust': customerId,
          'num': phoneNumber['number'],
          'type': phoneNumber['location']
      }
      cursor.execute(phoneInsert, phoneData)

    nnection.commit()
    rsor.close()
  return customerId
```

MongoDB

```
def addUser(database, user):
    return database.customers.insert_one(user).inserted_id
```

Fig. 10. Comparison of SQL and MongoDB code to insert a single user

- Authentication: MongoDB offers integration with external security mechanisms including LDAP, Windows Active Directory, Kerberos, and x.509 certificates.
- Authorization: Role-Based Access Controls (RBAC).
- Encryption: MongoDB data can be encrypted on the network, on disk and in backups. With the Encrypted storage engine , protection of data-at-rest is an integral feature within the database.

For workload isolation MongoDB supports Operational and analytic workloads isolation from one another on different replica set nodes, so that they never contend for resources. For instance, Replica set tags allow read operations to be directed to specific nodes within the cluster, providing physical isolation between analytics and operational queries [7].

Moreover, MongoDB offers a fully managed, on-demand and elastic service, called MongoDB Atlas, in the public cloud. Atlas enables customers to deploy, operate, and scale MongoDB databases on AWS, Azure, or GCP in just a few clicks or programmatic API calls.

B. ArrangoDB

ArangoDB is a transactional document store. Internally, data is stored in a binary form (VelocyPack, a fast and compact format for serialization and storage), but what is entered and returned from the system is typically JSON. User can choose between two storage engines with different characteristics as shown in fig.11.

MMFiles (memory-mapped files)	**RocksDB** (since v3.2)
default	optional
dataset needs to fit into memory	work with as much data as fits on disk
indexes in memory	hot set in memory, data and indexes on disk
slow restart due to index rebuilding	fast startup (no rebuilding of indexes)
volatile collections (only in memory, optional)	collection data always persisted
collection level locking (writes block reads)	Document level locks, concurrent reads and writes

Fig. 11. ArrangoDB storage engines

In a situation when schema validation is needed in ArrangoDB, it can be integrated by a JOI validation in a ArangoDB Foxx microservice. ”Foxx is a JavaScript framework for writing data-centric HTTP microservices” that is integrated within ArangoDB. Foxx is based on Google’s V8 engine (it is written in C++, it’s a open source high-performance JavaScript and WebAssembly engine), which is used in applications such as Google Chrome. Directly integrating Foxx into the ArangoDB core, ensures that the framework has full access to all core functions on a C++ level. Foxx framework allows application users to write their data access and domain logic as microservices, and operate directly within a database with native access to in-memory data [1].

ArrangoDB has its own specific query language called AQL as discussed in section VI results under query concepts. AQL is classified as DQL and DML, which means it can create, read, update and delete documents along with support for complex queries.

When ArangoDB is started in cluster mode, each instance can play different and multiple roles. They can be:

- Coordinators are stateless and responsible for query handling, query processing.
- DB Servers are stateful and where the data is actually stored.
- Agency is a highly available and resilient key/value store.
- Agents hold the current cluster configuration in the Agency.

These are some of the system specific optimizations offered by ArrangoDB system.

C. Couchbase

Couchbase is a distributed multi-model NoSQL document-oriented database , which is a merger between CouchDB and Membase to create an easily scalable and high performance database. Creating a new database in Couchbase is a simple process, with no overhead. In fact, its can be simply done by issuing a single HTTP request. Documents are stored in JSON format, allowing user to take advantage of JSON arrays and dictionaries to represent collections of data [2].

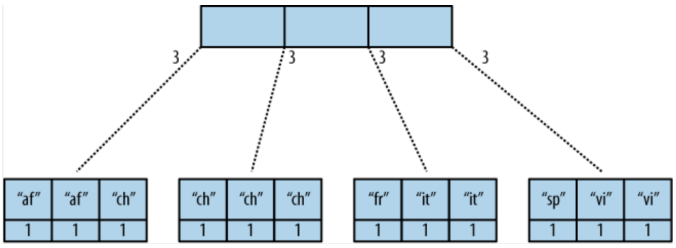

Fig. 12. B-tree in Couchbase

Couchbase has several benefits, including its efficient B-Tree data store implementation (refer Fig.12) and replication/synchronization support. Views in Couchbase are build using a map/reduce algorithm. When building a view, Couchbase will feed all new documents and the documents that have changed since the last time the view was built through a map function. The map function selects the documents of interest for that particular view. Following that a reduce function is run to calculate aggregate statistics on the documents that have been selected (min, max, count, sums, etc).

From Couchbase 4.0, developers have introduced N1QL, which is a powerful query language that extends SQL to JSON, enabling users to leverage both the power of SQL and the flexibility of JSON objects. It supports all the standard operations such as FROM, WHERE, SELECT statements, additionally, it also supports aggregation commands (such as GROUP BY), sorting (such as SORT BY), all types of joins (LEFT OUTER / INNER), as well as querying nested arrays and collections. Finally, query performance can be improved with composite, partial, covering indexes, and many more [**?**].

To protect who can read and update documents, Couchbase has a simple reader access and update validation privilege model that can be extended to implement custom security models. Administrator Access, Update Validation are couple of security functions provided by Couchbase.

- Couchbase database instances have administrator accounts. Administrator accounts can create other administrator accounts and update design documents.
- As documents are written to disk, they can be validated dynamically by JavaScript functions for both security and data validation. When the document passes all the formula validation criteria, the update is allowed to continue.

The Couchbase storage system treats edit conflicts as a common state, not an exceptional one. The conflict handling model is simple and while preserving single document semantics and allowing for decentralized conflict resolution. Couchbase allows for any number of conflicting documents to exist simultaneously in the database.

These are some of the system specific optimizations offered by Couchbase system.

VIII. Conclusion and Future Work

This literature review of document-oriented databases will reveal benefits of shifting from relational to NoSQL databases. Further this review reveals the differences between the architecture and policies of various document-oriented databases. The study has discussed different functionalities such as Database concept i.e. schemaless model, Data Modeling, Query Concepts, Scaling-Out and Data Locality, and Data Locality and Availability. This reveals the benefits of using document databases over traditional relational databases.

This is followed by discussion on system specific optimizations in three open-source databases namely: MongoDB, ArrangoDB, and Couchbase. This can help user to decide the better system based on their needs such as system security or data distribution etc.

In future I want to compare all-purpose open source document databases like MongoDB to optimized and proprietary systems like Amazon DynamoDB and Microsoft CosmosDB. We further want to benchmark these system to reveal what area of data storage are these systems fit to serve.

Acknowledgment

I would like to thank Dr. Paulo Alencar for his continuous support and feedback throughout the project.

References

[1] ArangoDB Inc. *Switching from Relational Databases to ArangoDB*. ArangoDB Inc., Aug 2018.

[2] CouchBase. *Why NoSQL? whitepaper*. Couchbase Inc.

[3] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou. Variability in software systemsa systematic literature review. *IEEE Transactions on Software Engineering*, 40(3):282–306, March 2014.

[4] Shady Hamouda and Zurinahni Zainol. Document-oriented data schema for relational database migration to nosql. In *2017 International Conference on Big Data Innovations and Applications (Innovate-Data)*, pages 43–50. IEEE, 2017.

[5] Girts Karnitis and Guntis Arnicans. Migration of relational database to document-oriented database: Structure denormalization and data transformation. In *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, pages 113–118. IEEE, 2015.

[6] KB Sundhara Kumar, S Mohanavalli, et al. A performance comparison of document oriented nosql databases. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–6. IEEE, 2017.

[7] MongoDB. *MongoDB Architecture Guide*. MongoDB, 2018.

[8] Lucas Olivera. Everything you need to know about nosql databases, Jun 2019.

[9] Zacharoula K. Papamitsiou and Anastasios A. Economides. Learning analytics and educational data mining in practice: A systematic literature review of empirical evidence. *Educational Technology Society*, 17:49–64, 2014.

[10] Dusan Petkovic. Json integration in relational database systems. *International Journal of Computer Applications*, 168:14–19, 06 2017.

[11] Jaroslav Pokorny. Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82, 2013.

[12] Harley Vera, Maristela Holanda Wagner Boaventura, Valeria Guimaraes, and Fernanda Hondo. Data modeling for nosql document-oriented databases. In *CEUR Workshop Proceedings*, volume 1478, pages 129–135, 2015.

[13] Lanjun Wang, Shuo Zhang, Juwei Shi, Limei Jiao, Oktie Hassanzadeh, Jia Zou, and Chen Wangz. Schema management for document stores. *Proceedings of the VLDB Endowment*, 8(9):922–933, 2015.