ENHANCING QUERY TIME USING A VOLUME-ADAPTIVE BIG DATA MODEL OF RELATIONAL DATABASES

Obilikwu, P[1]., Ogbuju, E.[2], & Kwaghtyo, K. D.[3]
[1,3]Department of Mathematics and Computer Science, Benue State University, Makurdi, Nigeria
[2]Department of Computer Science, Federal University, Lokoja, Nigeria
*Corresponding Author: emeka.ogbuju@fulokoja.edu.ng

Abstract
Big Data has been traditionally associated with distributed systems, the reason being that the volume dimension of Big Data, it appears, can be best accommodated by the continuous addition of inexpensive resources. It is within this implementation context that the non-distributed database models such as the relational database model have been faulted and departure from their usage contemplated by the database community. The atomicity, consistency, isolation, and durability (ACID) *properties of the relational database model however constitute a major attraction especially for applications that process transactions. A transaction-laden application may demand a lot more of the ACID properties of a database so as to maintain data integrity while requiring that the ever-increasing volume of data is also accommodated. This means that a one-size-fits-all database as proposed by several researchers may end up as a mirage and the current trend suggests that databases be made adaptive in the areas of their weakness rather than throw the baby away with the bath. This paper appreciates that the query time is negatively impacted as data volume increases in a relational database and therefore proposes a Big Data model of the relational database that partitions a relation thereby allowing volume to grow within partitions rather than a single relation. The results of the experiments performed show that the query time is enhanced as more data is accommodated in the partitions.*

Keywords: *Big Data, V-dimensions of data, adaptive model of relational DBMS, application prototypes, NoSQL, ACID properties*

1. Introduction

Relational databases have come under fire due to the varying dimensions data has taken without the relational model having ready answers. Storey and Song (2017) points out that, the relational database management system (RDMS) simply cannot handle Big Data. That is, data is too big, too fast, and too diverse to store and manipulate in RDMS as it requires a schema before writing to the database, a process which is assumed to be too rigid to handle the V-dimensions of Big Data. The ACID properties are also assumed to be too strict for some applications. This led to the requirements for new architectures and new transaction management techniques such as Basically Available, Soft State, Eventual consistency (BASE), which relaxes the ACID properties in distributed data management systems common in NoSQL systems (Storey and Song, 2017).

The NoSQL paradigm and its variants have also not provided a one-size-fits-all solution to the emerging database problems due to the shortcomings associated with the new paradigm (Stonebraker and Çetintemel, 2018). Incidentally, the shortcomings are in areas where the relational model has an edge. Storey and Song (2017) observed that the shift from relational databases to NoSQL database is spurred as well by the need for flexibility both in the scaling and data modeling. In terms of scaling in relational databases, scaling up is accomplished by adding a bigger server when additional capacity is needed. In NoSQL, scale-out means that, instead of acquiring a bigger server, one can add more commodity servers. NoSQL was specifically designed to address the needs of Big Data, and cloud computing. Activities important for dealing with Big Data issues include scalability, schema flexibility, ease of development, cost, and availability of deployment options. However, Storey and Song (2017) observed that most NoSQL databases lack full ACID compliance for guaranteeing transactional integrity and data consistency. Many NoSQL databases do not guarantee consistency, by design, because many applications need to handle potential inconsistencies. Eventually, consistency limits the use of NoSQL databases for mission-critical transactional applications.

Considering the two scenarios above, it appears a combination of several models will be a workable model for database applications so as to enable them accommodate the V-dimensions of data. Going deeper into the background of this study, the major Vs used to define the characteristics of Big Data are discussed.

1.1 Big Data and its V Characteristics

Hemlata and Gulia (2016) describes Big Data as a dataset which could not be captured, managed, and processed by general computers within an acceptable scope or time frame. Storey and Song (2017) characterized Big Data by the "3Vs" of volume, variety, and velocity, emerging from advances in sensing, measuring, and social computing technologies. In addition to these Vs, veracity and especially value are important and each of the Vs has its unique challenges (Storey and Song, 2017). Figure 1 summarizes the "5 V" challenges dominant in Big Data practice and research efforts.

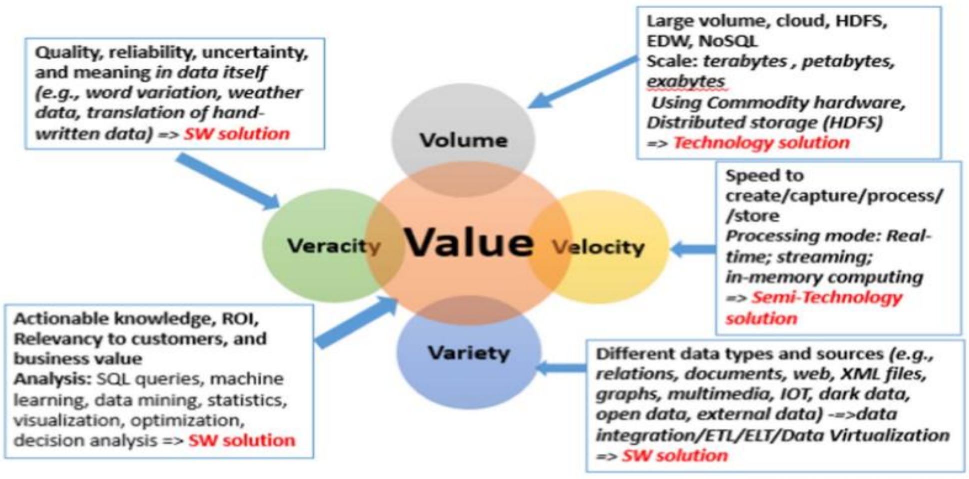
Quality, reliability, uncertainty, and meaning in data itself (e.g., word variation, weather data, translation of hand-written data) => SW solution
Large volume, cloud, HDFS, EDW, NoSQL
Scale: terabytes, petabytes, exabytes
Using Commodity hardware, Distributed storage (HDFS)
=> Technology solution
Volume
Veracity
Value
Velocity
Speed to create/capture/process/ /store
Processing mode: Real-time; streaming; in-memory computing
=> Semi-Technology solution
Actionable knowledge, ROI, Relevancy to customers, and business value
Analysis: SQL queries, machine learning, data mining, statistics, visualization, optimization, decision analysis => SW solution
Variety
Different data types and sources (e.g., relations, documents, web, XML files, graphs, multimedia, IOT, dark data, open data, external data) -->data integration/ETL/ELT/Data Virtualization
=> SW solution

Figure 1: The 5 Vs of Big Data (Storey and Song, 2017)

A. **Volume:** The volume of Big Data refers to the size of data being created from all the sources including text, audio, video, social networks, research studies, medical data, space images, crime reports, weather forecasting and natural disasters as defined by Khan, Uddin and Gupta (2014). The scale is now terabytes, petabytes, and exabytes. Storey and Song (2017) points out that the volume challenge is being addressed technologically by using commodity hardware and the Hadoop Distributed File System (HDFS).
B. **Velocity:** The velocity is the speed at which data is created, captured, extracted, processed, and stored. A semi-technology solution is needed to deal with the velocity challenge, with the software solution portion having real-time processing, streaming and in-memory computing (Storey and Song, 2017).
C. **Variety:** Variety connotes different data types and sources (from structured, semi-structured and unstructured data), documents, Web data, XML files, sensor data, multimedia files, and so forth. The variety challenge is primarily addressed by software solutions because the integration of heterogeneous data requires an extensive software effort to handle the variety (Storey and Song, 2017).
D. **Veracity:** Veracity means the truthfulness of data (Khan et al., 2014). The veracity refers to the accuracy of the data. It raises issues of quality, reliability, uncertainty, incompleteness, and the meaning in the data itself (e.g., word variation, weather data, and translation of hand-written data). Eventually, the veracity must be consistent to be processed in an automated manner and its challenge should be addressed by software solutions (Storey and Song, 2017).
E. **Value:** The value of Big Data refers to data relevancy to users (as evidenced by much research on text mining and sentiment analysis); and other measures. The needed analysis of Big Data to identify such value may occur in various ways including traditional SQL-type queries, machine learning techniques, data mining, statistics, optimization, and decision support analysis. The results may be represented in different forms, including traditional, standard and adhoc report generation, and visualization. The value challenge is most difficult to achieve as its software solutions must be addressed within the context of the business or problem domain. The next section exposes the three Vs used to define the characteristics of volume which is the cog of Big Data as defined by Patgiri and Ahmed (2016).

1.2 Volume Re-defined

Volume is the major dimension of Big Data such that if it is removed from the Vs of Big Data then Big Data becomes a small set of data which is well fitted within any conventional system to store and manage. Big Data and volume are analogous in meaning, and thus volume is an integral part of Big Data. Since volume is the backbone of Big Data, database technologies must support volume with capacity to store, process and manage large data sets. Volume is the main dilemma of Big Data that must be conquered and it will not be out of place to say that the probability of volume ceasing to be a requirement for Big Data will decrease soon is zero. This justifies this study and several others focused on the behaviour of volume. From the perspective of hardware, the floppy disk was changed to DVD to address volume and right now, the DVD is almost obsolete. In another dimension, the SSD is emerging to replace the HDD and different type of RAM technologies have been introduced. For instance, RAMCloud (Stonebraker and Çetintemel, 2018) has been introduced to overcome the latency issue. On the other hand, the software perspective is also promoting the technology transfer by innovating new technologies and devising new algorithm. For instance, erasure coding creates more empty storage spaces (Patgiri and Ahmed, 2016).

To put volume in a perspective that emphasizes its relevance to Big Data, volume is redefined by voluminosity, vacuum and vitality, three additional V-dimensions of data. Patgiri and Ahmed (2016) exposed that these 3V's define the characteristics of volume in Big Data as shown in Figure 2.

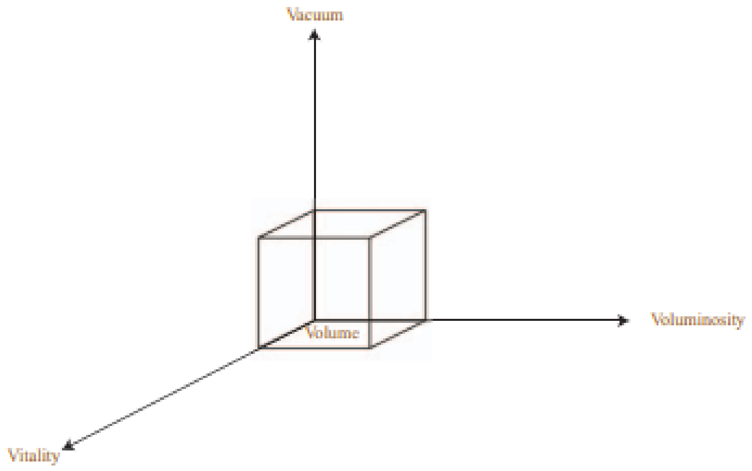

Figure 2: Re-definition of Volume of 3V's of Big Data (Patgiri and Ahmed, 2016)

A. **Voluminosity:** Voluminosity in volume states that there is a very large set of data collected so far and even much more is available to be collected. Of course, the volume collected so far and to be collected has a significant gap (Patgiri and Ahmed, 2016).
B. **Vacuum:** The vacuum in volume states that there is a strong requirement for empty spaces to store large volumes of data. Vacuum also refers to the creation of room to store, process and manage tremendous dataset from the existing datasets. This dimension of data pops up the research question about how much storage space is available for incoming data rather than how much data we have stored. The process of creating storage space for incoming data is equally as challenging as it is with managing vast sets of already stored data. Vacuum is concerned with creating space, either augmenting storage devices or other techniques to reduce the size of data (Patgiri and Ahmed, 2016).
C. **Vitality:** The vitality of volume states that there is a massive amount of data actively served and unserved. Vitality emphasises the survival of data in the storage environment and thus its reliability. In a large data bank, there is some data which are actively used why some are not (Patgiri and Ahmed, 2016). However, companies generate revenue from the actively used data only and the rest are stored in hope for future uses. There is the risk that data stored for future use is abandoned or not properly maintained as the risk gets higher, anything can happen to those data. In other words, with less investment and attention to the unserved data, they are exposed to incidences

of fire, earthquake, flood, war, and terrorist which are the prominent causes of data loss. Thus, vitality is a critical component of volume. In the absence of vitality, there will be no disaster management system and reliability will be lost or decimated. Apart from reliability, vitality also describes flexibility, dependability and security. Vitality is an integral component of volume just as volume is to Big Data.

In the related work, techniques and approaches proposed and some implemented to address the Big Data challenge exposed by the V-dimensions of data are discussed.

2. Related Works

Big Data has been presented in the literature by researchers as a departure from the relational database model. This mindset has given birth to newer database models and approaches such as the P-stores, C- stores, NoSQL and the S-stores among others that have been proposed by scholars to address the Big Data issue. This section exposes the current trends describing efforts made by researchers to take care of Big Data issues over the years.

2.1. Pre-NoSQL Stores

Pre-NoSQL stores include C-stores, P-stores and the S-stores among others. The root of column-oriented database systems often termed C-store can be traced to the 1970s. In recent years, some column store databases namely; MonetDB and C-Store have been introduced with the claim that their performance gains are quite noticeable against traditional approaches. The traditional approaches referred to are row-oriented database systems that have physical designs such that almost all the tables in the database have a one-to-one mapping to the tables in the logical schema (Yaman, 2012). The performance of the open-source C-store database shows that, although the internal structure of a column store is emulated inside a row store, the query processing performance of the C-store is quite poor (Yaman, 2012). An attempt to optimise the performance of C-stores led to the design of the P-store, another form of pre-NoSQL store.

P-Store is a partially replicated data store for wide-area networks developed by Schiper, Sutra, and Pedone that provides transactions with serializability (Ölveczky, 2017). Ölveczky (2017) observes that, P-Store executes transactions concurrently and that the execution of a transaction (T) at site (S) proceeds without worrying about conflicting concurrent transactions at other sites. P-Store assumes that the local executions of multiple transactions on a site are equivalent to some serialized executions. This assumption is modeled by executing the transactions one-by-one. Therefore, a replica can only receive a transaction request if it's set of currently executing transactions is empty. However, the certification protocol in P-store causes delays in transactions and the need for stream processing paved way to S-store technique.

S-Store, is a data management system that combines Online Transaction Processing (OLTP) transactions with stream processing (Cetintemel et al., 2014). S-Store belongs to a new breed of stream processing systems designed for high-throughput, scalable, and fault-tolerant processing over big and fast data across large clusters. Cetintemel et al. (2014) emphasized that S-Store is a client-server system and it is unique in that, all data access in S-Store is SQL-based and fully transactional. However, the inherent stream processing in S-Store exposes data and processing dependencies among transactions that are not captured by the model hence, the need for a better Big Data solution.

2.2. NoSQL and NewSQL

Moniruzzaman and Hossain (2013) exposed the fact that the acronym NoSQL was coined in 1998. Many people thought NoSQL is a derogatory term created to poke at SQL. In reality, the term, means Not Only SQL. The idea is that both technologies can coexist and each has its place. Over the years, many of the Web 2.0 leaders have adopted NoSQL technology. Companies like Facebook, Twitter, Digg, Amazon, LinkedIn and Google all use NoSQL in one way or another (Moniruzzaman and Hossain, 2013). According to Gessert et al. (2016), a group of data storage systems able to cope with Big Data are subsumed under the term NoSQL databases, which emerged as a backend to support Big Data applications. In recent years, the amount of useful data in some applications like social media, sensor networks have become so vast that it cannot be stored, processed or managed by the traditional database systems. Gandini et al. (2014) asserts that NoSQL databases are characterized by horizontal scalability, schema-free data models, and easy cloud deployment. They have capability to manage large amounts of data, hence they have become widely adopted on cloud platforms. The growing importance of Big Data applications (Moniruzzaman and Hossain, 2013; Gessert et al., 2016; Gandini et al., 2014) has driven the development of a wide variety of NoSQL databases such as Google's BigTable, Amazon's Dynamo, Facebook's Cassandra, and Oracle's NoSQL DB, MongoDB, Apache's HBase and others.

NewSQL (Storey and Song, 2017) is a class of new breed databases that have the strengths of both relational and NoSQL databases. They support SQL and take advantage of its ACID properties. They are built on the scale-out architecture, supporting scalability and fault tolerance. NewSQL databases provide a scalable performance comparable to NoSQL systems for OLTP workloads. However, NewSQL have limited support for "variety" due to the need of a schema. Google spanner, VoltDB, MemSQL, NuoDB and Clustrix are examples of databases based on the NewSQL database paradigm.

2.3 Hadoop and MapReduce

Map/Reduce is a programming paradigm with automatic parallelization. The Map part applies to the input data. It emits reduction keys and values with the output sorted and partitioned for the Reduce aspect. The Reduce function is applied to data grouped by the reduction key. The reduce function "reduces" data in the sense that it can aggregate data by adding selected values. The Map and Reduce operations are then chained together for complex computations. The result is extreme scalability, well-suited for scale-out architectures that use low-cost commodity hardware with fault-tolerant features (Storey and Song, 2017; Kumar et al., 2014). Hadoop can process and store large amounts of structured, unstructured and semi-structured data. Hadoop is an open-source version of the Map/Reduce algorithm, which was created to analyze large amounts of unstructured data and has become a de facto standard for Big Data. In a traditional database, a query is written in a structured query language, the data is accessed as stored in a relational database, and the result obtained (Das and Mohapatro, 2014). These types of queries, however, can be limited, so the desired output may not be obtained. Using Hadoop, unstructured data can be combined in many ways to facilitate data access. The Hadoop ecosystem progresses from data storage, data processing, and data access, to data management as

1. Data storage – HDFS (Hadoop distributed file system) and HBase (column database storage);
2. Data processing – MapReduce (automatic parallel data processing);
3. Data access – Hive (SQL-like); Pig (data flow); Mahout (machine learning); Avro (data serialization and remote procedure protocol); and Sqoop (relational database management connector).

Hadoop, as viewed also by Kumar et al. (2014) is inherently scalable and good for processing a large amount of data with automatic load balancing. Hadoop, however, is too dependent on HDFS when multiple iterations are needed and still requires significant manual coding to implement complex operations such as joins based on multiple fields. These limitations brought about a new memory-resident parallel processing framework, called Spark (Storey and Song, 2017).

2.3.1 Apache Spark

Apache Spark is an in-memory centric computing platform, designed specifically for large scale analytical processing. It is a fast and generic engine with a simple and expressive programming model for supporting a wide range of applications, including ETL (Extract, Transform, and Load), machine learning, stream processing, and graph computations. Eighty high-level operators make it easy to build parallel applications, with interactive use from Scala, Python and R shells (Storey and Song, 2017) . It combines SQL, streaming, and complex analytics. Spark has a stack of libraries that can be combined in a single application, and include SQL and Data Frames, MLlib for machine learning, GraphX, and Spark Streaming. Spark can access diverse data sources such as HDFS (Hadoop Distributed File Sharing), Cassandra (column-based database), HBase (Hadoop's database), Hive, and Tachyon. It uses Resilient Distributed Datasets (RDDs), which are fault-tolerant distributed memory abstractions that avoid replication. Spark can interactively query 1 to 2 terabytes of data in less than one second. Whereas Hadoop is good for batch applications, Spark is good for running real-time or iterative applications such as machine learning or graph processing and it is easier to program than Hadoop (Storey and Song, 2017).

Literature reviewed has exposed the fact that resolving the challenges associated with Big Data involves scaling the data store. Scalability can be vertical or horizontal.

2.4 Vertical and Horizontal Scaling

The pre-NoSQL and NoSQL stores and any attempt to make a database management system adaptive to the dimensions of data involves a partitioning scheme that makes it possible for the database to be scaled. Scaling, from the viewpoint of Tailor and Patel (2016), is the ability to handle increase or decrease in database storage demands.

Vertical Scaling (scaling up) means resource maximization of a single unit to increase its ability to handle the ever-increasing load. From the perspective of hardware, this includes adding memory and processing power to the physical machine on which the database server is running as shown in Figure 3. From the perspective of software or programming, scaling up may include optimizing application code and algorithms. Parallelizing or optimizing a number of running processes is also considered as methods of scaling up.

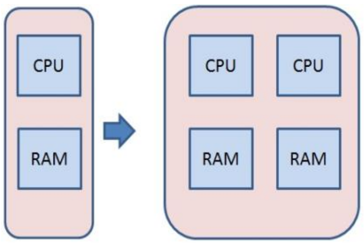

Figure 3: Vertical Scaling (Tailor and Patel, 2016)

Although scaling up may be relatively straightforward, the method suffers from several disadvantages. Initially, the addition of hardware resources reflects in decreasing returns and only increases as the additional resource is put to optimal use. Besides, there is the inevitable downtime needed for scaling up. If all of the web application services and data remain on a single unit, then vertical scaling on such unit does not give assurance on the application's availability (Tailor and Patel, 2016). This led to the idea of horizontal scaling.

Horizontal scaling (scaling out) refers to resource increment by the addition of units that work in unison with an existing system as depicted in Figure 4. This means the addition of more units of smaller capacity instead of the replacement of an existing single unit with one of larger capacity. Haven scaled out, data is then partitioned using a partitioning strategy and spread across multiple units or servers, hence, reducing the excess load on a single machine (Das and Mohapatro, 2014)

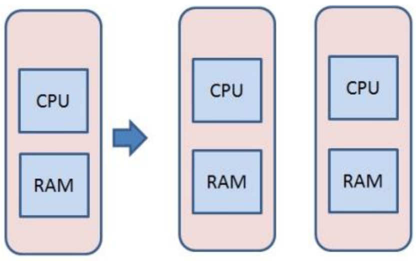

Figure 4: Horizontal Scaling (Tailor and Patel, 2016).

Having multiple units work together creates the positive probability of keeping the entire system up even if some of the units go down, thus, avoiding the “single point of failure” problem. In this way, horizontal scaling enhances the availability of the system. In addition, the aggregate cost incurred by numerous smaller machines is less than the cost of a single larger unit. Thus, it can be said that horizontal scaling can be more cost-effective in contrast to vertical scaling. Increasing the number of units implies that more resources need to be invested in maintenance. Additionally, the code of the application itself will need to work in such a way as to permit parallelism and distribution of work among various units. In some circumstances, this task is not trivial and scaling horizontally can become a tough task (Tailor and Patel, 2016).

Big Data applications need to run on numerous servers or Virtual Machine (VM) instances to manage user traffic (Tailor and Patel, 2016). Existing approaches to that effect are based on either vertical or horizontal scaling or partitioning. Tailor and Patel (2016) presented a comparison between horizontal scaling and vertical scaling (see Table 1), the two techniques of scaling cloud computing resources. The comparison considers complexity, throughput, cost and efficiency of both techniques.

Table 1 Comparative Analysis of Horizontal and Vertical Scaling (Tailor and Patel, 2016).

S/N	Indices	Vertical Scaling	Horizontal Scaling
1	Meaning	Increase the resources in the same logical unit or server aimed at increasing capacity	increasing the performance of a server or node by adding more instances of server to the pool of servers so as to spread workload
2	Reason to scale	It includes increasing IOP$_s$ (Input / Output Operations), increasing disk capacity and CPU/RAM capacity.	It includes increasing I/O concurrency, increasing disk capacity and reducing the load on existing nodes.
3	Efficiency	Vertical scaling is fairly inefficient regarding resource reallocation. Because servers are inclined to be dedicated to specific tasks, it can be tough to reallocate the spare resources of a server to other, more processing tasks.	Horizontal scaling, on the other hand, adds more nodes to the system as it scales, rather than it beefs up the existing nodes. This is relatively the more popular scaling strategy
4	Complexity	Less complex	More complex
5	Throughput	Less throughput	Horizontal scaling permits more throughputs.
6	Application/database server	Application server or database server is central	Each node has a separate application or database server
7	Failure Recovery	Failure recovery is a very difficult task	Failure Recovery is easy
8	Approach	Scale-up approach	Scale-out approach
9	Scenario	This scenario concentrates on the situation that a hardware platform has enough capacity to host more than one instance of an application. The application is reproduced on the same hardware until the capacity requirements are met.	This scenario means the ability to increase a system's capability or its performance by replicating a system (comprising of hardware or a virtualized platform) until the capacity requirement is satisfied.
10	Cost	Expensive	Cost-effective

Using either the vertical or horizontal scaling strategy, several approaches have been followed to deploy Big Data. Agrawal and Nyamful (2016) examine and summarize current storage technologies for Big Data applications. Variables such as capacity, scalability, data transfer rate, access time, and cost of storage devices, are highlighted in the study. Abourezq and Idrissi (2016) observed that one of the most adapted answers to Big Data storage requirements is cloud computing, and more specifically Database-as-a-Service (DBaaS), which allows storing and managing tremendous volume of variable data seamlessly, without need to make large investments in infrastructure, platform, software, and human resources. In this context, the article presents a benchmark of the main database solutions that are offered by service providers as a DBaaS and discussed their adaptability to Big Data applications. As it is the case with very many researchers in this area, this work did not capture in it any practical approach but only gave a benchmark of the main database solutions offered by service providers.

Xirogiannopoulos et al. (2017) proposed an end-to-end graph analysis framework called GraphGen, that subsumes the different design points where relational or graph data models or engines are combined. GraphGen is intended as a layer on top of a relational database, and although it can simulate the different design points, it does not, as of now, offer solutions to all of the optimization challenges that arise in the process. GraphGen considers graph analytics or querying as a combination of (1) specifying graphs of interest against the data in the underlying database as GraphViews, and (2) specifying an analysis task or a query (possibly at a later time) against those graphs. The study encountered several difficult challenges in terms of deciding where to execute graph queries/tasks, re-writing the SQL queries, and handling inaccuracies of the query optimizer and database statistics exposed by natural graph extraction and analysis tasks.

Paola et al. (2011) presented an approach to modeling data generated by a hybrid simulator for wireless sensor networks, where virtual nodes coexist with real ones to ease the design and testing phases of sensor applications controlling large sites, such as entire office buildings. In the paper, scalability is a fundamental requirement concerning the number of users and also to the number of sensory devices and the environments under observation. However, the approach was sensory-based and not generic to take care of the ever-increasing diverse Big Data in several other areas.

Garani (2010) proposed the Nested Relational Algebra (NRA) and a database model called Nested Relational Model (NRM) defined for nesting relations at arbitrary nesting levels. All the operators have been recursively defined. As a result, there is no need to flatten the nested relations when a series of operations are executed and so the data redundancy and duplications caused by un-nesting relations is avoided. Furthermore, the representation of the data is claimed to be in a “natural form”. Making it easier for users to understand when working with the data since even complex objects can be modelled in one relation. The incorporation of spatial data to NRM and lack of optimisation techniques for the efficient evaluation of complex queries became a major setback of the system.

Spoth et al. (2017) presented a vision for adaptive schema databases (ASDs), a conceptual framework for querying unstructured and semi-structured data through iteratively refined, personalized schemas. ASDs can be realized leveraging probabilistic query processing techniques by incorporating extraction and integration into the DBMS. ASDs have the potential to bridge the gap between relational databases and NoSQL, creating a far more user-friendly data exploration experience than either approach is capable of. The paper, however, represents only the first step towards practical ASDs. That is to say that, ASD was partially implemented hence the loss of its grounds to proffer solution to the alarming challenge of volume dimension of Big Data.

The comparative analysis shows that a one-size-fits-all approach to the design and implementation of data stores is an idea whose time has come and gone (Stonebraker and Çetintemel, 2018; Idreos and Kraska, 2019; Pwint and Zhaoshun, 2019). Lunguand Mihalache (2016) therefore discussed a shift from a traditional static data approach to a more adaptive model approach to database design. The adaptive approach helps organizations build dynamic capabilities to react in a dynamic environment. Moniruzzaman and Hossain (2014) points to the fact that the traditional RDBMS can be complemented by specifically designing a rich set of alternative DBMS; such as NoSQL, NewSQL and Search-based systems and not a total departure from the status quo. It is on the basis of this new trend that this paper takes a more pragmatic view of Big Data implementations to mean the extent to which a database management system can accommodate some or all of the *V*-dimensions of data. While some of the database management systems have inherent properties that accommodate some of the V-dimensions of data, such database management systems can be made adaptive to some or all of the other *V*-dimensions of data. In the context of related work reviewed, this study argues that the relational database can be made adaptive to volume so to tackle the volume challenge through partitioning a relation. While some database product vendors may be individually taking steps in this direction, the problem of standardization looms large thereby making a study such as this imperative.

3.0 Materials and Methods

Handling Big Data is a very big issue in relational databases and the solution was and will always be scalability. Incidentally, relational databases scale vertically although with the emergence of New SQL, there is the hope of scaling relational databases horizontally. The point of this paper is that vertical scalability creates room for data to grow relations. However, retrieving a record or a set of records from a relation is done relative to the number of the total number of records in the relation. Based on this relationship, query time can be computed as a ratio using equation 1.

$$q_t = \frac{T_r}{T_R} \quad \ldots \qquad (1)$$

where q_t is query time, T_r is number of tuples retrieved from a relation *R and* T_R is number of tuples in R

The implication of Equation 1 is that an increase in query time comes with increase in volume afforded relational databases by vertical scalability. In this work, partitioning is done within the context of vertical scalability and it is shown that not only is more data accommodated but query time reduces as the data retrieval ratio shown in equation 1 is reduced by partitioning. The proposed adaptive relational database model takes care of volume and at the same time enhances query time by partitioning a relation using an appropriate attribute or set of attributes as partition keys. The value set of a partition key must not have a null value. The partition keys are then used to form the partition predicates on the basis of which the database instances are created.

Equation 1 is validated by the experimental results of this work based on the experimental data set in Table 2. As a running example throughout the paper, consider the relation (R) in Table 2 as storing information about students in a university:

Table 2: Relation (R)

Tuple	Matno	Dept	CourseCode	Session	Level	Location
T_1	...	MC	...	17/18	...	NW
T_2	...	BIO	...	16/17	...	SS
T_3	...	MC	...	18/19	...	NE
T_4	...	CHM	...	15/16	...	NC
T_5	...	MC	...	17/18	...	SE
T_6	...	CHM	...	16/17	...	NC

Assuming that the cardinality of R denoted as Card(R) has become large such that queries on R denoted as Q(R) have become very slow. Partioning R solves the associated volume problem and the department (dept) attribute qualifies eminently for use as a partition key. The distinct values in the value set associated with the partition key are MC, BIO and CHM. The distinct values produce three partition predicates, namely Dept= “MC”, Dept= “BIO” and Dept= “CHM”. The partition predicates are SARGable predicates (Selinger et al., 1979) and hence they filter the tuples of relation R into three partitions. The cardinality of relation R, Card(R) is 6 and the partition predicates identified partition R as follows:

Let P_1 = partition by Dept=MC,
then Card (P_1) = 3
P_2 = partition by Dept=BIO,
then Card (P_2) = 1
P_3 = partition by Dept=CHM,
then Card (P_3) = 2

The partitions P_1, P_2 and P_3 are relations and can be named MC, BIO and CHM respectively. Once a choice of a partition key is made, the partition predicates are automatically determined using the distinct values of the value set associated with the partition key. This is done dynamically and at run time hence the model is said to be adaptive. At all times, the cardinality of a partition is less than the cardinality of R implying volume is optimised. Again, this implies that the model is adaptive to volume. Generalising, it can be said for any relation (R) given a set of partition keys, A, then R = $\{P_1, P_2, ..., P_n\}$ where n = the number of distinct values in the value set associated with a partition key. The number of distinct values in the value set is also the number of partitions produced. That R = $\{P_1, P_2, ..., P_n\}$ is demonstrated by the proof of concept below.

3.1 Proof of Concept

Equation 1 shows that the larger the number of tuples in a relation, the longer the query time. To reduce query time, the number of records can be reduced by partitioning the records into smaller number of tuples. This approach to improving query time is proved using the following theorem and axiom:

Theorem: Given P_1, P_2 ... P_n as the partitions of a relation R, then R = {P_1, P_2, ..., P_n} where *n* = the number of distinct values in the value set associated with the partition key that generated P_1, P_2 ... P_n

Axiom: The following axioms are applicable:

1. A partition key has a value set, V whose element cannot be null
2. The number of distinct values of V is *n*= number of partitions produced

Proof: Let **6** be the partition predicate associated with a distinct value of V, then Card(**6**) is the cardinality of the tuples filtered by **6**.

Given any value of *n*, there exists 6_1, 6_2, ..., 6_n, where
6_1 filters all tuples in P_1 from relation R,
6_2 filters all tuples in P_2 from relation R, and
6_n filters all tuples in P_n from relation R,
Since the elements of V cannot be null, then Card(V) = Card (R)
Since 6_1, 6_2, ..., 6_n filter the tuples of R according to the distinct values of V, it follows that

$$\text{Card(V)=Card}(6_1) + \text{Card}(6_2) + ... + \text{Card}(6_n)=\sum_i^n Card(\sigma_i)$$

This implies that $\sum_i^n Card(\sigma_i)$= Card (R) since *n* is the number of distinct values of V defined in R

This shows that R = {P_1, P_2, ..., P_n} since 6_1, 6_2, ..., 6_n filter the tuples of R. **QED**.

The proof shows that partitioning does not change the data set. The experimental results go a step further to show that partitioning enhances query time.

4. Results and Discussion

The equivalence of a relation and its partitions according to a partition key has been demonstrated thereby proving that partitioning makes the relation adaptive to volume. The implementation of this adaptive model attempted in this section demonstrates the partitioning scheme discussed as well demonstrate empirically that the queries on the partition, 6_1, 6_2, ..., 6_n has a better query time compared to an equivalent query, *6* on the original relation, say R. The implementation considers a relation of students' registration details with a representative sample of the records shown in Table 3:

Table 3: Students' registration details fully populated in relation (R)

Tuple	Matno	Dept	Course Code	Session	Level	Location
T_1	32224	MC	CMP 422	17/18	400	NW
T_2	23433	BIO	ZOO 342	16/17	300	SS
T_3	55466	MC	MTH 341	18/19	300	SE
T_4	77656	CHM	CHM 141	15/16	100	NC
T_5	33266	MC	STAT 431	16/17	200	SW
T_6	99877	CHM	CHM 211	18/19	200	NE
T_7	32242	MC	CMP 322	15/16	100	NW
T_8	27833	BIO	ZOO 342	16/17	300	SE
T_9	55277	MC	MTH 311	18/19	200	NE
T_{10}	88656	CHM	CHM 141	15/16	100	SS
T_{11}	39966	MC	STAT 411	17/18	300	SE
T_{12}	91179	CHM	CHM 211	16/17	200	NW
⋮	⋮	⋮	⋮	⋮	⋮	⋮

As earlier exposed in the literature, partitioning is achieved only if there exist a relation, hence, for the purpose of this research, the following code snippet or create statement automatically creates a relation if it does not exist:

```
function createStudentTb($costudy){
// called by dossierplus/student.php
//echo " costudy = ".$costudy;
$tablename="dept ".RemoveSpecialChar($costudy);
if(!checkRemBigdataTableExist("student" . RemoveSpecialChar($costudy))) {
$query2="CREATE TABLE IF NOT EXISTS ".addslashes($tablename) ."(
`id` varchar(45) default NULL,
`matno` varchar(45) default NULL,
`dept` varchar(200) default NULL,
`courseCode` varchar(200) default NULL,
`session` varchar(200) default NULL,
`level` varchar(200) default NULL,
`location` varchar(200) default NULL,
timestamped varchar(15) default NULL,
timedescription varchar(25) default NULL,
`status` varchar(45) default NULL,
PRIMARY KEY  (`regnocostermid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1";
$create=query($query2)or die(mysql_error()."error function createStudentTb");
} // if($create==1) return true; else return false;
} // end of function createStudentattendanceTb
function RemoveSpecialChar($value){$result  = preg_replace('/[^a-zA-Z0-9_ -
]/s','',$value);

return $result;
}
```

```
// create a big table if none exists by calling the function
createStudentTb($costudy);
```

A select statement implements a partition predicate that fetches the records from the original relation and inserts them in the appropriate partitioned relation. This dynamically creates or splits the original relation based on an appropriate partition predicate. Following is the code snippet to perform the dynamic portioning of a relation:

```
// generate relation dynamically
$depttbname=" student ".RemoveSpecialChar($costudy);
$classtbname=" class ".RemoveSpecialChar($costudy);

// select records based on the partitioning rule and then insert inappropriate partition

$insertq = query("INSERT INTO $depttbname (`tuple` , `matno`, `dept`, `courseCode`,
`level`, `location`, `timestamped`, `timedescription`, `status`)
VALUES ('$id', 'Kenn', 'bsu32242', '17/18', 'two', 'NC'") or die(mysql_error("Error1"));
```

The query implements the partition predicates, Dept=”MC”, Dept=”BIO” and Dept=” CHM”, which in turn produce three partitions as shown in Table 4, 5 and 6

Table 4: MC_Dept (partition 1)

Tuple	Matno	Course Code	Session	Level	Location
T_1	32224	CMP 422	17/18	400	NW
T_3	55466	MTH 341	18/19	300	SE
T_5	33266	STAT 431	16/17	200	SW
T_7	32242	CMP 322	15/16	100	NW
T_9	55277	MTH 311	18/19	200	NE
T_{11}	39966	STAT 411	17/18	300	SE

Table 5: CHM_Dept (partition 2)

Tuple	Matno	Course Code	Session	Level	Location
T_4	77656	CHM 141	15/16	100	NC
T_6	99877	CHM 211	18/19	200	NE
T_{10}	88656	CHM 141	15/16	100	SS
T_{12}	91179	CHM 211	16/17	200	NW

Table 6: BIO_Dept (partition 3)

Tuple	Matno	Course Code	Session	Level	Location
T_2	23433	ZOO 342	16/17	300	SS
T_8	27833	ZOO 342	16/17	300	SE

The cardinality of the original relation is 12. The cardinality partitions 1, 2 and 3 are 6, 4 and 2 showing that $\sum_i^n Card(\sigma_i)$= Card (R). The partitions created can be hosted on a server or across different servers and by doing so, the volume dimension of Big Data is taken care of which is the primary concern in this work. Any other attribute of the original relations whose value set goes not have null elements can be used as a partition key. The guide is that the cardinality of the partitions produce must be such that query times on them are tolerable. The number of partitions produce must also be such that is manageable and not unwieldy. This means the choice of the partition key depends on the number of distinct values in its value set since that determines the number of partitions. Two or more partition keys can also be combined to produce a composite partition key and a corresponding conjunctive equality partition predicate.

Experimental queries applied on the original relation compared to the same queries applied on the partitions produced very interesting results. The partition predicates that partitioned the original relation were timed in each case. The same predicates were applied to fetch the records from the partitions respectively and note taken of the query times. Details of the experiments are shown in Table 7.

Table 7: Details of Experiments

Description	Predicate	Partition
Experiment 1	Dept="MC"	1
Experiment 2	Dept=" CHM",	2
Experiment 3	Dept="BIO"	3

The query time of each of the partitions is compared with the query time obtained when the same predicate is applied in a query on the original relation. The comparative results of the experiments are depicted in depicted in Figures 5, 6 and 7.

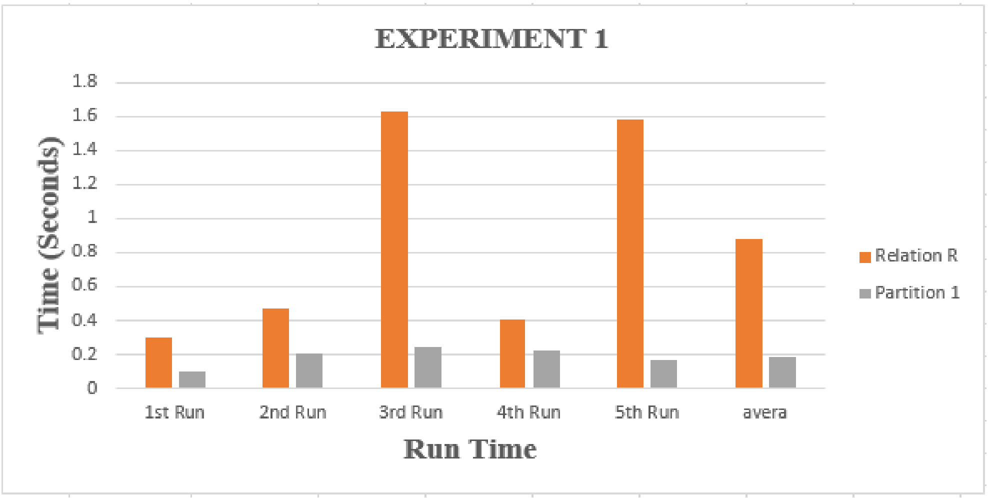

Figure 5: Relation (R) and partition (1) query time graph

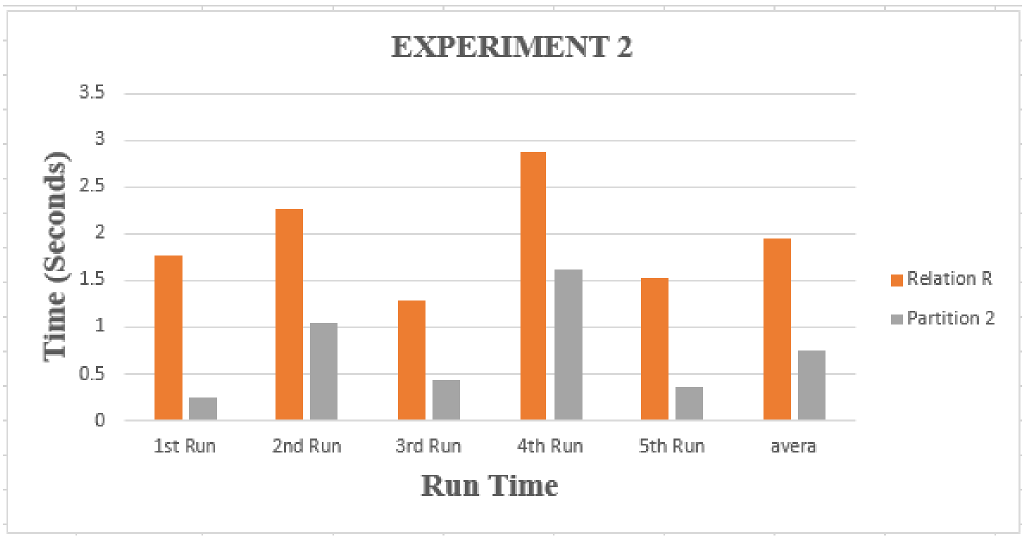

Figure 6: Relation (R) and partition (2) query time graph

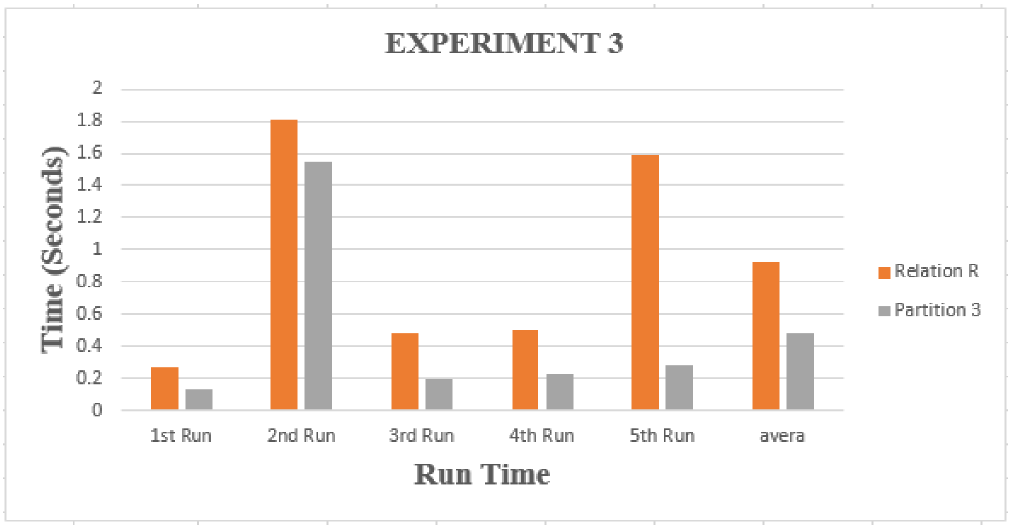

Figure 7: Relation (R) and partition (3) query time graph

It is observed from these results that in each run of experiment 1, 2 and 3, the query time associated with the query on each partition is less the query time associated with the query on the original relation. The difference is that the original ratio has more records than the partitions queried. This goes to prove empirically that equation 1 is true for relations and their corresponding partitions.

The ratios of the differences in query times across the runs of each experiment differ as shown the difference in the heights of the bar charts in the runs. This is attributed to "noise" factors in the run time environment. In an empirical query time analysis such as this, the influence of operating system processes on query time cannot be ruled out. However, the influence of the noise factors did not affect the big picture as the graph of the query times associated with the partitions are consistently lower than that of the query times associated with the original partition throughout all the runs of the experiments. This volume-adaptive model of a relational database has not only contributed in taking care of the volume dimension of relational data, but also optimised query time.

5. Conclusion

It has been shown that partitioning a relation in the relational database makes it adaptive to the volume requirement of Big Data. The adaptive model of the relational database also optimises query time as experimental results of this study shows. Unlike the schemaless model of Big Data, the adaptive model of Big Data produces partitions that are of uniform format thereby reducing the effort required to mine the data into a data lake platform for data analytics. The ACID-properties inherent in the relational database model are also preserved in the partitions. To improve the query time of partitions, a new partition predicate can always be implemented and the tuples of the partitions re-arranged accordingly. Making the model adaptive to the other *V*-dimensions of Big Data is being considered as part of future work. Accordingly, part of the future work would include establishing a hardware cluster architecture that would meet up with the horizontal scaling concept discussed in this work to cater for the holistic volume requirements.

References

Abourezq, M., & Idrissi, A. (2016). Database-as-a-Service for Big Data: An overview. *International Journal of Advanced Computer Science and Applications*, 7(1): 157-177

Agrawal, R., & Nyamful, C. (2016). Challenges of Big Data storage and management. *Global Journal of Information Technology*, 6(1): 01-10.

Cetintemel, U., Tufte, K., Wang, H., Zdonik, S., Du, J., Kraska, T., Maier, D., Meehan, J., Pavlo, A., Stonebraker, M., Sutherland, E., Madden, S., & Bitim, E. (2014). S-Store: A streaming NewSQL system for big velocity applications 7. Retrieved from http://www.cs.cmu.edu/~pavlo/static/papers/sstore_vldb14.pdf

Das, T.K. & Mohapatro, A. (2014). A study on Big Data integration with data warehouse. *International Journal of Computer Trends and Technology, 9(1):* 188-192.

Gandini A., Gribaudo M., Knottenbelt W.J., Osman R., Piazzolla P. (2014) Performance Evaluation of NoSQL Databases. In: Horváth A., Wolter K. (Eds.). (2014). *Lecture Notes in Computer Science: Vol 8721. Computer Performance Engineering*. Springer, Cham. https://doi.org/10.1007/978-3-319-10885-8_2

Garani, G. (2010). A generalized relational data model. *World Academy of Science, Engineering and Technology. 39.*

Gessert, F., Wingerath, W., Friedrich, S., & Ritter, N. (2016). NoSQL database systems: A survey and decision guidance. *Computer Science - Research and Development.* 10.1007/s00450-016-0334-3

Idreos, S., & Kraska, T. (2019). From Auto-tuning One Size Fits All to Self-designed and Learned Data-intensive Systems In 2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands. ACM, New York, NY, USA, 1-6. https://doi.org/10.1145/3299869.3314034

Kumar, R., Parashar, B.B., Gupta, S., Sharma, Y., & Gupta, N. (2014). Apache Hadoop, NoSQL and NewSQL Solutions of Big Data. *International Journal of Advance Foundation and Research in Science & Engineering 1*(6): 28-36

Khan, M.A., Uddin, M., & Gupta, N. (2014). Seven V's of Big Data understanding Big Data to extract value. *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, 1-5.

Lungu, I., & Mihalache, A. (2016). A new approach to adaptive data models. *Database Systems Journal* 7 (2): 19-27

Moniruzzaman, A.B.M & Hossain, S. (2013). NoSQL database: New era of databases for Big Data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Applications, 6(4): 1-14*

Ölveczky, P. (2017). Formalizing and validating the p-store replicated data store in Maude. 10.1007/978-3-319-72044-9_13

Patgiri, R., and Ahmed, A. (2016). Big Data: The V's of the game changer paradigm. IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and System, 17-24

Paola, D., Alessandra, L.R.G., Milazzo, F., & Ortolani, M. (2011). Adaptable data models for scalable Ambient Intelligence scenarios. *International Conference on Information Networking 2011,* 80 - 85. 10.1109/ICOIN.2011.5723138.

Pwint, P. K., & Zhaoshun, W. (2019). A review of polyglot persistence in the Big Data world. *Information 10*(4): 141; doi:10.3390/info10040141

Storey, V.C., and Song, I. (2017). Big Data technologies and Management: What conceptual modelling can do? *Data & Knowledge Engineering, 108(1): 50–67*

Stonebraker, M., & Çetintemel, U. (2018). One size fits all: an idea whose time has come and gone. In Proceedings of the International Conference on Data Engineering (ICDE), 2-11. 10.1145/3226595.3226636.

Spoth, W., Arab, B. S., Chan, E. S., Gawlick, D., Ghoneimy, A., Glavic, B., Hammerschmidt, B., Kennedy, O., Lee, S., Liu, Z. H., Niu, X., & Yang, Y... *Adaptive Schema Databases. CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. Retrieved from https://par.nsf.gov/biblio/10048275.

Selinger, P. G, Astrahan, M.M, Chamberlin, D.D, Lorie, R.A, Price, T.G (1979). Access Path Selection in a Relational Database Management System. *SIGMOD Conference 1979*, Boston, Massachusetts, May 30 - June 01, 23-34.

Tailor, U. & Patel, P. (2016). A survey on comparative analysis of horizontal scaling and vertical scaling of cloud computing resources, 2 (6): 2395-1052.

Xirogiannopoulos, K., Srinivas, V. & Deshpande, A. (2017). GraphGen: Adaptive graph processing using relational databases. 1-7. 10.1145/3078447.3078456.

Yaman, S.G. (2012). Introduction to Column-Oriented Database Systems. Columnar Databases, 2012. Retrieved from https://www.cs.helsinki.fi/webfm_send/1006/Sezin_final.pdf