ISSN 2278-3091
Volume 10, No.3, May - June 2021
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse1121032021.pdf
https://doi.org/10.30534/ijatcse/2021/1131032021

Volume-Adaptive Big Data Model for Relational Databases

Patrick O. Obilikwu[1*], Kenneth D. Kwaghtyo[1] and Edward N. Udo[2]
[1]Department of Mathematics and Computer Science, Benue State University, Makurdi, Nigeria
[2]Department of Computer Science, University of Uyo, Uyo, Nigeria.
[*]Corresponding author: poblikwu@gmail.com

ABSTRACT
Big data is traditionally associated with distributed systems and this is understandable given that the volume dimension of Big Data appears to be best accommodated by the continuous addition of resources over a distributed network rather than the continuous upgrade of a central storage resource. Based on this implementation context, non-distributed relational database models are considered volume-inefficient and a departure from their usage contemplated by the database community. Distributed systems depend on data partitioning to determine chunks of related data and where in storage they can be accommodated. In existing Database Management Systems (DBMS), data partitioning is automated which in the opinion of this paper does not give the best results since partitioning is an NP-hard problem in terms of algorithmic time complexity. The NP-hardness is shown to be reduced by a partitioning strategy that relies on the discretion of the programmer which is more effective and flexible though requires extra coding effort. NP-hard problems are solved more effectively by a combination of discretion rather than full automation. In this paper, the partitioning process is reviewed and a programmer-based partitioning strategy implemented for an application with a relational DBMS backend. By doing this, the relational DBMS is made adaptive in the volume dimension of big data. The ACID properties (atomicity, consistency, isolation, and durability) of the relational database model which constitutes a major attraction especially for applications that process transactions is thus harnessed. On a more general note, the results of this research suggest that databases can be made adaptive in the areas of their weaknesses as a one-size-fits-all database management system may no longer be feasible.

Key words: Big Data, V-dimensions of data, Adaptive Model of Relational DBMS, NoSQL, ACID properties

1. INTRODUCTION

Big data is described as a dataset that cannot be captured, managed, and processed by average-sized computers within an acceptable scope or time frame [1]. Several databases have continued to emerge in response to the many big data dimensions in which data now occur; a phenomenon for which the relational database model does not have ready answers. [2] pointed out that, the relational database management system (RDMS) simply cannot handle big data. That is, big data is too big, too fast, and too diverse to store and manipulate in RDMS because RDMS 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 (atomicity, consistency, isolation, and durability) of the relational database 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 BASE (Basically Available, Soft State, Eventual consistency), which relaxes the ACID properties in distributed data management systems such as the NoSQL paradigm [2].

Other issues that must be addressed when handling big data include scalability, schema flexibility, and ease of development, cost, and availability of deployment options. It is observed in [2] that the shift from relational databases to NoSQL database was spurred by the need for flexibility both in the scaling and data modelling possibilities required by big data. 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. However, NoSQL databases do not guarantee consistency, by design, because many applications need to handle potential inconsistencies. Eventually, lack of consistency limits the use of NoSQL databases for mission-critical transactional applications. Scaling up of relational databases is accomplished by adding a bigger server when additional capacity is needed. This is very expensive considering the capital outlay required to acquire new servers. [3] examined and exposed the state-of-the-art storage technologies for big data applications. Variables such as capacity, scalability, data transfer rate, access time, and cost of storage devices, are re-emphasized in the study to handle big data issues

To understand completely the requirements of big data beyond its requirement for storage capacity, the V-dimensions of big data are discussed. [2] 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 3Vs, other Vs such as veracity and value have been added.

Figure 1 summarizes the "5Vs" challenges dominant in big data practice and research efforts.

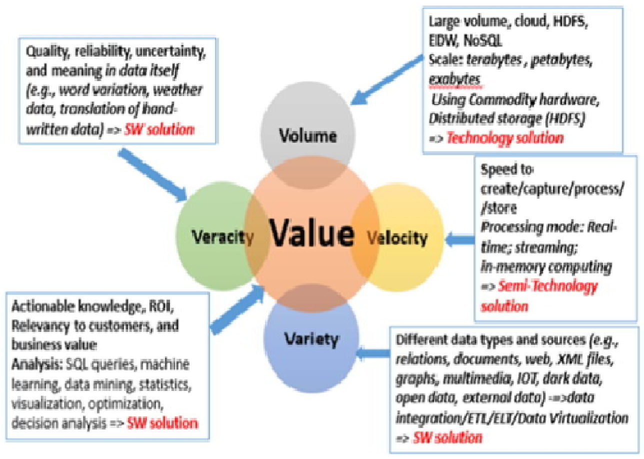
Quality, reliability, uncertainty, and meaning in data itself (e.g., word variation, weather data, translation of hand-written data) => SW solution
Large volume, cloud, HDFS, EDW, NoSQL
Scale: terabytes , petabytes, exabytes
Using Commodity hardware, Distributed storage (HDFS)
=> Technology solution
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 5Vs of Big Data [2]

The 5Vs are explained as follows:

i. **Volume:** This is the size of data being created from all the sources including text, audio, video, social networks, research studies, medical data, space images, crime reports, etc. as defined [4]. The scale is now in terabytes, petabytes, and exabytes. The volume challenge is being addressed technologically by using commodity hardware and the Hadoop Distributed File System (HDFS) [2].
ii. **Velocity:** Velocity is seen as the speed at which data is created, captured, extracted, processed, or 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 [2].
iii. **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 [2].
iv. **Veracity:** Veracity means the truthfulness of data [4]. Veracity refers to the accuracy of the data. It raises issues of quality, reliability, uncertainty, incompleteness, as well as 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 with the help of software solutions [2].
v. **Value:** Value is concerned with data relevance 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 ad-hoc report generation, and visualization. The value challenge is most difficult to achieve as its software solutions must be addressed within its context. The next section exposes the three Vs used to define the characteristics of volume which is the cog of Big Data as defined by [5]

The volume dimension of big data has been achieved mainly by distributing data in chunks to storage nodes. The nodes could be a data centre provisioned with a network of computers in physical proximity or data centres across the globe connected via the internet. This paper is aimed at reviewing how partitioning techniques are used to arrive at chunks of related data and exposing the crucial role of partitioning in achieving data distribution. The objectives of this paper are therefore threefold: (i) introduce big data in terms of the V-dimensions of data occurrence; (ii) review data partitioning and the fact that current database management systems implement automatic partitioning strategy; (3) demonstrate how the discretion of the developer in determining the partitioning strategy reduces the NP-hardness of the automated partitioning strategy.

The rest of this paper is organized as follows: Section 2 reviews related work necessary to better understand the technical aspects of this work, with an emphasis on the automated partitioning strategy. Section 3 describes the details of our proposed non-automated approach to partition design and implementation. The partition design is embedded in the database design and shown using appropriate design methodologies. In Section 4, it is shown how the relational database model can be adapted using non-automated partition strategies in its design to accommodate big data in its volume dimension. Section 5 concludes the paper and outlines current and future lines of research.

2. RELATED WORK

Approaches to handling big data have been exposed in the literature by researchers in response to the departure from the traditional pattern in which data presents itself in the monolithic forms of texts and digits. This departure has led 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 dimensions of data. This section unveils 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 database like MonetDB has been introduced with the claim that their performance gains are quite noticeable against traditional approaches. The traditional approaches 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 a logical schema [6]. The performance of C-store databases 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 [6]. 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 [7]. P-Store executes transactions concurrently and that the execution of a transaction (T) at the site (S) proceeds without worrying about conflicting concurrent transactions at other sites [7]. P-Store assumes that the local executions of multiple transactions on a site are equivalent to some serialized executions. This assumption is modelled by executing the transactions one-by-one. Therefore, a replica can only receive a transaction request if its set of currently executing transactions are empty. However, the certification protocol in P-store causes delays in transactions and the need for stream processing paved the way to the S-store technique.

S-Store is a data management system that combines Online Transaction Processing (OLTP) transactions with stream processing [8]. 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. [8] emphasized that S-Store is a client-server system and 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 solution like NoSQL.

2.2 **NoSQL and NewSQL**

The acronym NoSQL was coined in 1998 [9]. At first, 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, companies like Facebook, Twitter, Digg, Amazon, LinkedIn and Google adopted the use of NoSQL in one way or another [9]. According to [10], 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 has become so vast that it cannot be stored, processed or managed by the traditional database systems. [11] asserts that NoSQL databases are characterized by horizontal scalability, schema-free data models, and easy cloud deployment. They have capabilities to manage large amounts of data, hence become widely adopted on cloud platforms. The growing importance of big data applications 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 is a class of new breed databases that have the strengths of both relational and NoSQL databases[2]. 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 has limited support for "variety" due to the need for 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 can be seen as the input part. It houses the 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 reduced function aggregates 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 [2, 12]. 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, 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 is stored in a relational database, and the result obtained [13]. These types of queries, however, can be limited, thus the desired output may not be obtained. Using Hadoop, unstructured data can be combined in many ways to facilitate data access.

Hadoop 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 by [12], 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 [2].

2.3.1 **Apache Spark**

Apache Spark is a memory-centric computing platform, designed specifically for large scale 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 [2]. 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 [2].

2.4 **Volume Re-defined**

If the volume dimension of big data is absent from the properties of a dataset, then the data set becomes small even if has the other dimensions of big data. Volume is therefore the major dimension of big data. Big data and volume are synonymous in meaning, thus volume is a pillar that cannot be ignored in defining big data. Since volume is a pillar of big data, database technologies must support volume with the capacity to store, process and manage large data sets. Volume is the main issue in 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 data 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 types of RAM technologies have been introduced. For instance, RAMCloud [15] has been introduced to overcome the latency issue. On the other hand, the software perspective is also promoting big data technologies such as single-sign applications with big data backends [5]. A big data model supporting a single-sign-on application that enhances data comparability across multiple organisations was designed and implemented by [16]

To put volume in a perspective that emphasises its relevance to big data, volume is redefined by voluminosity, vacuum and vitality, three additional V-dimensions of data. These 3V's define the characteristics of volume in big data and are explained thus:

i. **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. The volume collected so far and to be collected has a significant gap [5].
ii. **Vacuum -** In volume, vacuum means there is a strong requirement for empty spaces to store large volumes of data. The vacuum also refers to the creation of room to store, process and manage the ever-emerging dataset. This dimension reiterates the research question that 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 challenging as with managing already stored data. Vacuum is therefore concerned with creating space or devising techniques to reduce the size of data [5].
iii. **Vitality -** The vitality of volume is concerned with the amount of data actively served and unserved. Vitality emphasises the survival of data and its reliability in the storage environment. In a large data store, some data are actively used why some are not [5]. However, companies generate revenue from the actively used data only and the rest are stored in the hope for future

uses. The tendency that the data stored for future use is abandoned or not properly maintained as the tendency increases, anything can happen to such data. Again, poor investment/attention to the unserved data leads to incidences of fire, earthquake, flood, war, and terrorism 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. Apart from reliability, vitality also describes flexibility, dependability and security. Vitality is an integral component of volume just as the volume is to Big data.

2.5 Adapting Data Stores for Volume

The problem of volume has been solved traditionally by upgrading hardware to accommodate more data in one central storage facility or data centre. With positive research results in distributed systems, it has become feasible to increase storage by simply adding inexpensive servers to an array of servers in a data centre or facilities outside the operational location via communication networks, At the heart of distributed data systems is the fact that data must be partitioned according to well-defined conditions that are used to identify them during storage and retrieval. Doing this achieves database scalability. The pre-NoSQL distributed systems and NoSQL stores scale by default, an ability to handle an increase or decrease in database storage demands [14]. While some scale vertically, others scale horizontally.

Vertical Scaling (scaling-up) deals with the resource maximization of a single unit to increase its ability to handle the ever-increasing load. From the hardware perspective, this includes adding memory and processing power to the physical machine on which the database server is running as shown in Figure 2. From the perspective of software or programming, scaling up include optimizing application code and algorithms. Parallelizing or optimizing several running processes is also considered methods of scaling up.

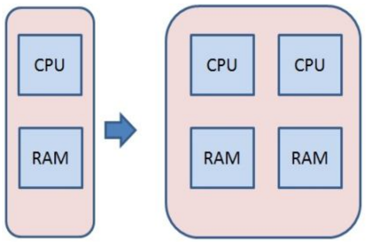

Figure 2: Vertical Scaling [14]

Although scaling up may be relatively straightforward, the method suffers from several disadvantages. Initially, the addition of hardware resources reflects decreasing returns and only increases as the additional resource is put to optimal use. Besides, there is an 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 a unit does not give assurance on the application's availability [14]. This led to the idea of horizontal scaling.

Horizontal scaling (scaling out) refers to the additional resources that work in unison with an existing system as depicted in Figure 3. 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, excess load on a single machine is reduced [13].

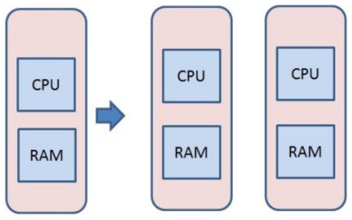

Figure 3: Horizontal Scaling [14]

Having multiple units working together creates the positive probability of keeping the entire system up even if some of the units go down. This avoids a single point failure problem. This way, horizontal scaling increases the availability of a system. Besides, the aggregate cost incurred for numerous smaller units is less than the cost of a single larger unit. That is, horizontal scaling minimizes cost when compared to vertical scaling. However, increasing the number of units implies that more resources are needed to be invested in maintenance. Also, the code itself needs to be compiled such that it can permit parallelism and distribution of work among various units. In some circumstances, this task is not trivial therefore scaling horizontally can be a tough task [14].

Big data applications require numerous servers or Virtual Machine (VM) instances to manage user traffic [14]. Existing approaches to that effect are based on either vertical or horizontal scaling or partitioning. [14] examined the following comparisons between horizontal scaling and vertical scaling, the two techniques of scaling cloud computing resources. The comparison considers the complexity, throughput, cost and efficiency of both techniques and is summarized in Table 1.

Table 1. Comparative Analysis of Horizontal and Vertical Scaling [14].

S/ N		**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 a server to the pool of servers to spread the 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 in terms of resource sharing. Because servers are dedicated to specific tasks. Hence, tough to share the spare resources with a more fast processing server.	Horizontal scaling, on the other hand, adds more nodes to the system as it scales, rather than beef up the existing nodes. This is relatively the more powerful and popular scaling strategy
4	Complexity	Less complex	More complex
5	Throughput	Less throughput	More throughputs
6	Application/database server	Has a centralized application or database server	Has a decentralized application or database server
7	Failure Recovery	Failure recovery is difficult	Failure Recovery is easy
8	Approach	Scale-up approach	Scale-out approach
9	Scenario	This scenario focuses on increasing the capability of a hardware platform 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 focuses on enhancing the capacity of a system or its performance through replication of a system (comprising of hardware or a virtual platform) until the capacity requirement is satisfied.
10	Cost	Expensive	Cost-effective

Using either the vertical or horizontal scaling strategy, the following approaches have been employed by researchers to deploy big data. [17] implemented one of the most adapted answers to big data storage known as Cloud Computing. More specifically, Database as a Service, which allows storing and managing a tremendous volume of variable data seamlessly, without the need to make large investments in infrastructure, platform, software, and human resources. [18] 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 (i) specifying graphs of interest against the data in the underlying database as GraphViews, and (ii) specifying an analysis task or a query (possibly at a later time) against those graphs. The study encountered 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. In [19], an approach for modelling data generated by a hybrid simulator for wireless sensor networks, where virtual nodes coexist with real ones is proposed to ease the design and testing phases of sensor applications controlling large sites, such as entire office buildings. In the study, scalability is a fundamental requirement concerning the number of users and also 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. [20] proposed the Nested Relational Algebra (NRA) and a database model called Nested Relational Model (NRM) for nesting relations arbitrarily. 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. The incorporation of spatial data to NRM and the lack of optimization techniques for the efficient evaluation of complex queries became a major setback of the system. Also, [21] proposed an adaptive schema database (ASD), a conceptual framework for querying unstructured and semi-structured data t iteratively. ASD leveraged the probabilistic query processing techniques by incorporating extraction and integration into the DBMS. It has the potential to bridge the gap between relational databases and NoSQL, creating a far more user-friendly data exploration experience. However, ASD has partially implemented hence the loss of its grounds to proffer a solution to the alarming challenge of the volume dimensions of big data.

It is obvious that both the NoSQL and relational database models each have their advantages and challenges. It appears a combination of several models will be a workable model for database applications to enable them to accommodate the V-dimensions of data. Alternatively, a database model can be tweaked in its area of weakness to enable it accommodate data in the V-dimensions of big data. In other words, the days of a one-size fits all are gone and in this paper, partitioning the relational database to accommodate big data is considered [15]. The partition strategy proposed relaxed the NP-hardness of automated partitioning using the discretion of the programmer in the application codes.

Automated Partitioning of Relational Database

The NoSQL database paradigm seems to have solved the problems associated with the V-dimensions data assumes when cultivated in large quantities about entities of different categories and most times different sources. But they have failed to guarantee the ACID properties of data as the failure of network node in a distributed system may mean the unavailability of a queried dataset among other challenges. A transaction-laden application may demand a lot more of the ACID properties of a database 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 the proponents of the NoSQL paradigm may end up as a mirage. It is obvious that the query time is negatively affected as data volume increases in a relational database and therefore proposes a big data model which partitions a relation in a relational database. The programmed partitions allow data to grow in the new partitions rather than a single relation. The fact that partitions contain less data than their non-partitioned equivalent enhances query time, hence another contributing factor considered in this study.

The ACID properties of the relational database model have been assumed to be too strict for some applications. The 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. In other words, a one-size-fits-all approach to the design and implementation of data stores is an idea whose time has come and gone [15, 22 - 24]. It is on the basis of this that several researchers have proposed different approaches to the design and implementation of database management systems. [14] for example, discussed a shift from a traditional static data approach to a more adaptive model approach to database design. The adaptive approach help organizations build dynamic capabilities to react in a dynamic environment. At present, leaving partitioning in the hands of the developer is one of such adaptive approaches to database design and implementation. However, [9] noted that the traditional RDBMS can be complemented by specifically designing a rich set of alternative DBMS; such as NoSQL, NewSQL and Search-based systems but not a total departure from the traditional RDBMS. It is based on this new trend that this paper takes a more pragmatic view of big data implementations

To make the relational database model adaptive to the volume dimension of big data, the trend is to build them with automatic partitioning features [26 - 28]. Automatic partitioning creates partitioning configurations based on which the database is broken down into chunks and distributed. The partitioning configuration is stored on a single machine as if it were a regular database (but with no actual but meta data). The partitioning configuration constitutes a search space of heuristics like genetic- and rank-based techniques that are used to identify units of data distributed. In its basic form, the heuristics make use of interesting columns [26]. Interesting columns represent an extension of the notion of interesting orders introduced in System R [22]. To arrive at a partitioning strategy, the following may be considered as interesting columns: (i) columns referenced in equality join predicates, and (ii) any subset of group-by columns. Join columns are interesting because they make local and directed joins possible. The group-by columns are interesting because aggregations can be done locally at each node and then combined. The interesting columns are by definition considered as partitioning candidates.

Automatic partitioning takes the control of the partitioning process from the programmer in addition to several other drawbacks of automatic partitioning. The study by [28] affirm that the main pitfall or challenge of self-regulated database partitioning is the fact enumerating the search space so as to arrive at an applicable partitioning strategy makes partitioning an NP-hard problem and leaving it to automation will not produce the best of results in terms of the resulting algorithmic time complexity. A partitioning strategy developed and implemented by the application developer is therefore proposed as being more effective though requiring extra coding effort.

3. METHODOLOGY

The effectiveness of the distributed database management system is based on the fact that query performance is enhanced if the cardinality of a relationship is minimal.. In a database D, a query time T, and a storage limit L, there exist a condition for D such that (i) the size of replicated (distributed) tables fits into L, and (ii) the overall query time T is minimized [26]. That is to say that the time taken to retrieve a record or a set of records from a relation is proportional to the total number of records in the relation hence the rationale for partitioning since it reduces the number of records in a unit relation. Based on this relationship, query time can be computed as a ratio using equation 1.

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

Where q_t is query time, T_r is the number of tuples retrieved from a relation R using a predicate σ and T_R is the number of tuples in R.

The implication of equation 1 is that an increase in query time comes with an increase in volume. In this study, partitioning is done within the context of horizontal scalability and has proven 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 the interesting columns. In the experimental scenarios in this study, the interesting columns are suggested in the database design and the design does not in any way depend on the database engine for implementation.

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	C.Code	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 relation R is denoted as Card (R), becomes large, then queries on relation R denoted as Q(R) become very slow. Portioning the relation 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 [26] 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 a run time hence the model is said to be adaptive. At all times, the cardinality of a partition is less than the cardinality of relation R, by implication, volume is optimised. Again, this implies that the model is adaptive to volume. Generally, it can be said that for any relation (R), given a set of partition keys, 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. The abstraction R = {P_1, P_2… P_n} is demonstrated by the proof of concept in section 3.1.

3.1 **Prof of Concept**

Equation 1 shows that the larger the number of tuples in a relation, the longer the query time. To address this delay in query time, the number of records can be reduced by partitioning the records into a 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 **σ** be the partition predicate associated with a distinct value V, then Card (**σ**) is the cardinality of the tuples filtered by **σ**.
Given any value of n, there exists σ_1, σ_2... σ_n,
Where:
σ_1 filters all tuples in P_1 from the relation R,
σ_2 filters all tuples in P_2 from the relation R, and
σ_n filters all tuples in P_n from the relation R,

Since the elements of V cannot be null, then Card (V) = Card (R)

Since σ_1, σ_2... σ_n filter the tuples of R according to the distinct values of V, it follows that:
Card (V) =Card (σ_1) + Card (σ_2) +.... + Card (σ_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 σ_1, σ_2... σ_n filter the tuples of R.

The proof shows that partitioning does not change the data set. The experimental results further demonstrate that partitioning enhances query time.

The equivalence of a relation and its partitions according to a partition key has been demonstrated. This study in its approach and contribution, further demonstrate how partitioning makes a relation or a database adaptive to volume. The implementation of this adaptive model thus demonstrates the partitioning scheme discussed as well demonstrate empirically that the queries on the partition, σ_1, σ_2, ..., σ_n have a better query time compared to an equivalent query, σ 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′ course 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	32224	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}	32242	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 that partitioning is achieved only if there exist a relation. Hence, for the purpose of this research, the following code snippet or create statement allows the programmer to create partitions from relation R. Relation is partitioned into chunks of data related by course of study indicated by the variable, $costudy:

```
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);
```

Bearing in mind that the main contribution of this study is to give programmer the privilege to create partitions as the need arises, a *select* statement is employed here to implement the partitioning predicate. This extracts records from the original relation and inserts them into the appropriately partitioned relation. In this way, the original relation is dynamically created or split based on an appropriate partitioning predicate. Following is a call to the function, createStudentTb($costudy), defined above to perform the dynamic portioning of relation:

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

// select records based on the partitioning rule and then
insert them into an appropriate 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"));
```

This query implements the partition predicates, Dept=”MC”, Dept=”BIO” and Dept=” CHM”, on user demand. It produces 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	32224	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 of partitions 1, 2 and 3 are 6, 4 and 2 respectively. This affirms that $\sum_i^n Card(\sigma_i) =$ Card (R). The partitions created can be hosted on a server or across different servers. Thus, the volume dimension of big data is taken care of which is the primary concern in this study. Any other attribute of the original relations whose value set does 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 produced must also be such that it is manageable and not unwieldy or cumbersome. This means that 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 partitioning predicate.

4. RESULT AND DISCUSSION

Experimentally, a relation R populated with 300,000 tuples or records was created. The relation R was then partitioned into three relations: 1, 2 and 3 of different sizes each. Out of these partitions, relation 1 comprises 150,000 tuples, relation 2 consists of 60,000 tuples and relation 3 is made of 90,000 tuples. The number of tuples in the three partitioned relations amounts to 300,000 tuples as contained in the original relation R. A query was applied on the original relation R with each of the partitions produced from relation R and the resulting query time was taken. That is, the same predicate on the partitioned relations and the original relation were timed in each case. Details of the experiments are as shown in Table 7.

Table 7: Details of Experiments

Description	**Experiment 1**		**Experiment 2**		**Experiment 3**	
Run Time	**Relation R**	**Partition 1**	**Relation R**	**Partition 2**	**Relation R**	**Partition 3**
1st Run	0.304412	0.100755	1.757856	0.248396	0.265892	0.133841
2nd Run	0.46711	0.204732	2.256918	1.04493	1.809146	1.548764
3rd Run	1.634166	0.238915	1.285841	0.431768	0.481732	0.197544
4th Run	0.405944	0.221573	2.87495	1.626576	0.499015	0.232302
5th Run	1.584439	0.167413	1.532954	0.358075	1.58721	0.281298
Ave_RunTime	0.8792142	0.1866776	1.9417038	0.741949	0.928599	0.4787498

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 graphically demonstrated in Figures 4, 5 and 6

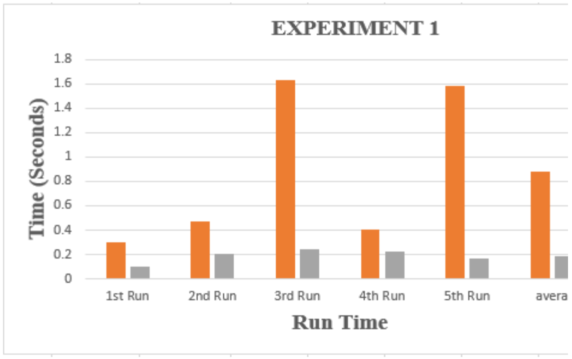

Figure 4: Relation (R) and partition (1) query time graph

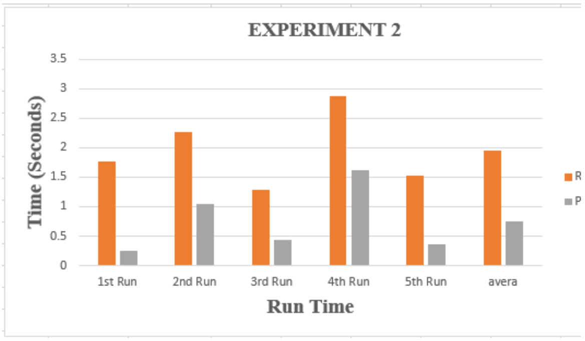

Figure 5: Relation (R) and partition (2) query time graph

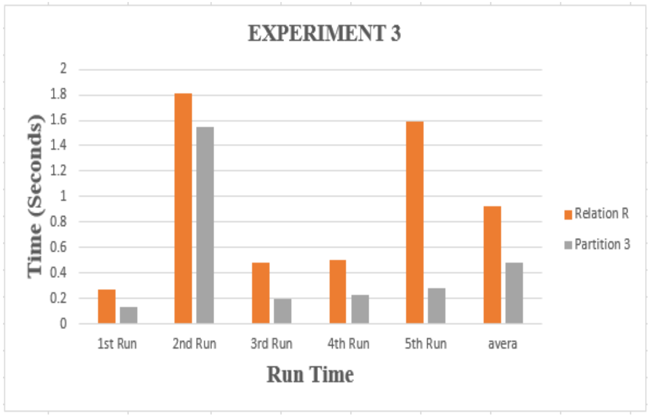

Figure 6: Relation (R) and partition (3) query time graph

The partitions in the experiments were produced by partition predicates embedded in the codes written in function, createStudentTb($costudy).It is observed from these results that in each run of experiment 1, 2 and 3, the query time of the partitioned relation is less than the query time of the original relation R in Table 3. This is because relation R has more records than the partitioned relations in each case. This shows that the non-automatic strategy for partitioning is effective and produces the established result that query time is enhanced when data is partitioned. This result goes far to affirm empirically that equation 1 is true for relation R and its corresponding partitions.

The ratio of the differences in query times across the runs of each experiment differs as evident in the heights of the bar charts in the various run times displayed in the graphs. 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 to taking care of the volume dimension of relational data, but also optimizes query time.

5. CONCLUSION AND FUTURE WORK

It is practically demonstrated in this study that using codes representing the discretion of the programmer, a relational database can be partitioned and the expected result that query time is improved achieved. This is preferred to automated partitioning because automated partitioning is based on algorithms and heuristics whose algorithmic time complexity is NP-Hard. The partition strategy implemented in this study makes it possible to store voluminous data across several partitions thereby makes the relational database adaptive to the volume requirement of big data. Unlike the schemaless model of big data, the adaptive model of the relational database 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 also include distributing the partitions across a hardware cluster architecture using the horizontal scaling concept earlier discussed in the related work section to cater for the holistic volume requirements of big data.

REFERENCES

[1] C. Hemlata and P. Gulia. **Big Data Analytics**, Journal of Computer and Information Technology Sciences, vol. 4, no. 2, pp. 1 – 4, 2016

[2] V. C. Storey and I. Song. **Big Data Technologies and Management: What Conceptual Modelling can do**, Data & Knowledge Engineering vol. 108, pp. 50–67, 2017

[3] R. Agrawal and C. Nyamful. **Challenges of Big Data Storage and Management,** Global Journal of Information Technology, vol. 6, no. 1, pp. 1-10, 2016

[4] M. A. Khan, M. F. Uddin, and N. Guptam. **Seven V's of Big Data: Understanding Big Data to Extract Value,** In Proceedings of 2014 Zone 1 Conference of the American Society for Engineering Education (ASEE Zone 1), Bridgeport, Connecticut, USA, 2014

[5] R. Patgiri and A. Ahmed. **Big Data: The V's of the Game Changer Paradigm**, In Proceedings of 18th IEEE International Conference on High-Performance Computing and Communications, Sydney, Australia. 2016

[6] S. G. Yaman, **Introduction to Column-Oriented Database Systems,** Seminar: Columnar Databases**,** Finland, 2012

[7] Peter Ölveczky, **Formalizing and Validating the P-Store Replicated Data Store in Maude**, In book: Recent Trends in Algebraic Development Techniques, doi: 10.1007/978-3-319-72044-9_13, 2017

[8] Ugur Cetintemel, Kristin Tufte, Hao Wang, Stanley Zdonik, Jiang Du, Tim Kraska, et al.**, S-store: A Streaming NewSQL System For Big Velocity Applications**. In Proceedings of the VLDB Endowment, vol 7, Issue 13, pp. 1633–1636, 2014

[9] A. B. Moniruzzaman and Syed Hossain, **NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison**, International Journal of Database Theory and Application, vol. 6, no. 4, pp. 1 – 13, 2013

[10] Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter. **NoSQL Database Systems: A Survey and Decision Guidance**, Computer Science Research and Development, vol. 32, Issue 3-4, pp 353–365, 2017

[11] Gandini A., Gribaudo M., Knottenbelt W.J., Osman R., Piazzolla P. **Performance Evaluation of NoSQL Databases**. In: Horváth A., Wolter K. (eds) Computer Performance Engineering. EPEW. Lecture Notes in Computer Science, vol 8721, Springer, Cham, 2014

[12] Rakesh Kumar, Bhanu Parashar, Sakshi Gupta, Yougeshwary Sharma, and Neha Gupta. **Apache Hadoop, NoSQL and NewSQL Solutions of Big Data**, International Journal of Advance Foundation and Research in Science & Engineering, vol. 1, issue 6, pp. 28 – 36, 2014

[13] T. K. Das and Arati Mohapatro. **A Study on Big Data Integration with Data Warehouse**. International Journal of Computer Trends and Technology. Vol. 9, pp. 188-192, 2014

[14] U. Tailor and P. Patel. **A Survey on Comparative Analysis of Horizontal Scaling and Vertical Scaling of Cloud Computing Resources**, International Journal for Science and Advance Research in Technology, vol. 2, issue 6, pp. 2395-1052, 2016.

[15] Michael Stonebraker and Ugur Çetintemel. **One Size Fits All: An Idea Whose Time has come and gone**, In book: Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker, 10.1145/3226595.3226636, 2018

[16] Patrick Obilikwu and Emeka Ogbuju. **A Data Model for Enhanced Data Comparability across Multiple Organizations**., Journal of Big Data, vol. 7, article no. 95, 2020

[17] Manar Abourezq and Abdellah Idrissi. **Database-as-a-Service for Big Data: An Overview**. International Journal of Advanced Computer Science and Applications, vol. 7, no. 1, pp. 157 – 177, 2016

[18] Konstantinos Xirogiannopoulos, Virinchi Srinivas and Amol Deshpande. **GraphGen: Adaptive Graph Processing using Relational Databases**. In Proceedings of the Fifth International Workshop on Graph Data-management Experiences and Systems, Article No. 9, pp. 1 – 7, 2017

[19] Alessandra De Paola, Giuseppe Lo Re, Fabrizio Milazzo,Marco Ortolani. **Adaptable Data Models for Scalable Ambient Intelligence Scenarios**. In Proceedings of International Conference on Information Networking, Kuala Lumpur, Malaysia, pp. 80 – 85, 2011

[20] Georgia Garani. **A Generalised Relational Data Model.** International Journal of Computer Systems Science and Engineering, vol. 4, no. 1, pp. 43 – 59, 2010

[21] W. Spoth, B. S. Arab, E. S. Chan, D. Gawlick, A. Ghoneimy, B. Glavic, et al.. **Adaptive Schema Databases**. In Proceedings of 8th Biennial Conference on Innovative Data Systems Research (CIDR ’17). Chaminade, California, USA, 2017

[22] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie and T. G. Price. **Access Path Selection in a Relational Database Management System,** SIGMOD Conference, Boston, Massachusetts, pp. 23-34, 1979

[23] S. Idreos and T. Kraska. **From Auto-tuning One Size Fits All to Self-designed and Learned Data-Intensive Systems**, SIGMOD Conference, Amsterdam, Netherlands, 2019

[24] Pwint P. Khine. and Zhaoshun Wang. **A Review of Polyglot Persistence in the Big Data World**, Information, vol. 10, no. 4, doi: 10.3390/info10040141, 2019

[25] Ion Lungu and Andrei Mihalache**. A New Approach to Adaptive Data Models**. Database Systems Journal, vol. 7, no. 2, pp. 19 – 27, 2016.

[26] Rimma Nehme and Nicolas Bruno. **Automated Partitioning Design in Parallel Database Systems**, In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, Athens, Greece, pp. 1137-1148, 2011

[27] Alvin Cheung, Owen Arden, S. Madden and Andrew C. Myers. **Automatic partitioning of database applications**, In Proceedings of the VLDB Endowment, Vol. 5, Issue 11, pp. 1471 –1482, doi: 10.14778/2350229.2350262, 2012

[28] Sanjay Agrawal, Vivek Narasayya and Beverly Yang. **Integrating Vertical and Horizontal Partitioning into Automated physical database design**, In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, pp. 359 – 370, 2004.