



## Storage Solutions for Big Data Systems: A Qualitative Study and Comparison

Khan, Samiya; Liu, Xiufeng; Ali, Syed Arshad; Alam, Mansaf

*Published in:*  
ArXiv

*Publication date:*  
2019

*Document Version*  
Early version, also known as pre-print

[Link back to DTU Orbit](#)

*Citation (APA):*

Khan, S., Liu, X., Ali, S. A., & Alam, M. (2019). Storage Solutions for Big Data Systems: A Qualitative Study and Comparison. *ArXiv*, (arXiv:1904.11498). <https://arxiv.org/abs/1904.11498>

---

### General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

# Storage Solutions for Big Data Systems: A Qualitative Study and Comparison

Samiya Khan<sup>a,1</sup>, Xiufeng Liu<sup>b</sup>, Syed Arshad Ali<sup>a</sup>, Mansaf Alam<sup>a,2</sup>

<sup>a</sup>Jamia Millia Islamia, New Delhi, India

<sup>b</sup>Technical University of Denmark, Denmark

---

## Highlights

- Provides a classification of NoSQL solutions on the basis of their supported data model, which may be data-oriented, graph, key-value or wide-column.
  - Performs feature analysis of 80 NoSQL solutions in view of technology selection criteria for big data systems along with a cumulative evaluation of appropriate and inappropriate use cases for each of the data model.
  - Classifies big data file formats into five categories namely text-based, row-based, column-based, in-memory and data storage services.
  - Compares available data file formats, analyzing benefits and shortcomings, and use cases for each data file format.
  - Evaluates the challenges associated with shift of next-generation big data storage towards decentralized storage and blockchain technologies.
- 

## Abstract

Big data systems' development is full of challenges in view of the variety of application areas and domains that this technology promises to serve. Typically, fundamental design decisions involved in big data systems' design include choosing appropriate storage and computing infrastructures. In this age of heterogeneous systems that integrate different technologies for optimized solution to a specific real-world problem, big data system are not an exception to any such rule. As far as the storage aspect of any big data system is concerned, the primary facet in this regard is a storage infrastructure and NoSQL seems to be the right technology that fulfills its requirements. However, every big data application has variable data characteristics and thus, the corresponding data fits into a different data model. This paper presents feature and use-case analysis and comparison of the four main data models namely document-oriented, key-value, graph and wide-column. Moreover, a feature analysis of 80 NoSQL solutions has been provided, elaborating on the criteria and points that a developer must consider while making a possible choice. Typically, big data storage needs to communicate with the execution engine and other processing and visualization technologies to create a comprehensive solution. This brings forth second facet of big data storage, big data file formats, into picture. The second half of the research paper compares the advantages, shortcomings and possible use cases of available big data file formats for Hadoop, which is the foundation for most big data computing technologies. Decentralized storage and blockchain are seen as the next generation of big data storage and its challenges and future prospects have also been discussed.

**Keywords:** Big Data Storage, NoSQL, Big Data File Formats, Decentralized Storage, Blockchain

---

---

Corresponding authors.

E-mail addresses: <sup>1</sup>[samiyashaukat@yahoo.com](mailto:samiyashaukat@yahoo.com) (Samiya Khan), <sup>2</sup>[malam2@jmi.ac.in](mailto:malam2@jmi.ac.in) (Mansaf Alam)

## 1.0 Introduction

Data is processed to generate information, which can later be used for varied purposes. Data mining and knowledge discovery are two fields that have been actively working towards deriving useful information from raw data to create applications that can make predictions, identify patterns and facilitate decision making [1]. However, with the rise of social media and smart devices, data is no longer a simple dataset that traditional tools and technologies can handle [2].

Digitization and rising popularity of modern technologies like smart phones and gadgets has contributed immensely towards ‘data deluge’. Moreover, this data is not just high on volume, but it also includes data of varied kinds that is generated on a periodic basis. The biggest challenge in dealing with this ‘big data problem’ is that the present or traditional systems are unable to store and process data of this kind. Therefore, this gave rise to the need for scalable systems that can store varied forms of data and process the same to generate useful analytical solutions [3].

The present era can rightly be called the era of analytics where organizations are tapping the business potential of data by processing and analyzing it. A plethora of technologies are available for this purpose and organizations are smoothly drifting towards heterogeneous environments, which include data stores like HBase [4], HDFS [5] and MongoDB [6], execution engines like Impala [7] and Spark [8], and programming languages like R [9] and Python [10].

Big data storage [11] is a general term used for describing storage infrastructures designed for storage, management and retrieval of data that is typically large in volume, high in velocity and diverse in variety. In such infrastructures, data is stored in such a manner that its usage, processing and access become easier. Moreover, such infrastructures can scale as per the requirement of the application or service.

The primary task of big data storage is to support input and output operations on stored data in addition to storage of a large number of files and objects. Typically, the architectures used for storage of big data include a cluster of network-attached storage, pools of direct attached storage or storage based on object storage format [11]. Computing server nodes are used at the heart of these infrastructures in order to provide support for retrieval and processing of big data. Most of these storage infrastructures provide support for big data storage solutions like Hadoop [12] and NoSQL [13].

The storage needs of a big data problem are influenced by many factors. Fig. 1 recapitulates the requirements from a big data storage system, which include –

- Scalability  
Scalability is undoubtedly one of the fundamental requirements in view of the ever-growing size of data. Any solution made for big data must be able to accommodate the growing data in an optimal manner.
- Availability  
Considering the fact that most big data solutions require real-time analysis of data and its visualization, the allowable time in which data must be accessed is extremely low. Moreover, data needs to be accessed in a frequent and efficient manner, making availability a crucial system requirement.
- Security  
Big data solutions may make use of organization-specific data. One of the biggest concerns of organizations in the adoption of such solutions is the security of their data.
- Integration  
There may be a need for big data solutions to interact with other technologies and applications. Therefore, big data solutions must be able to integrate with these solutions to create a complete application.

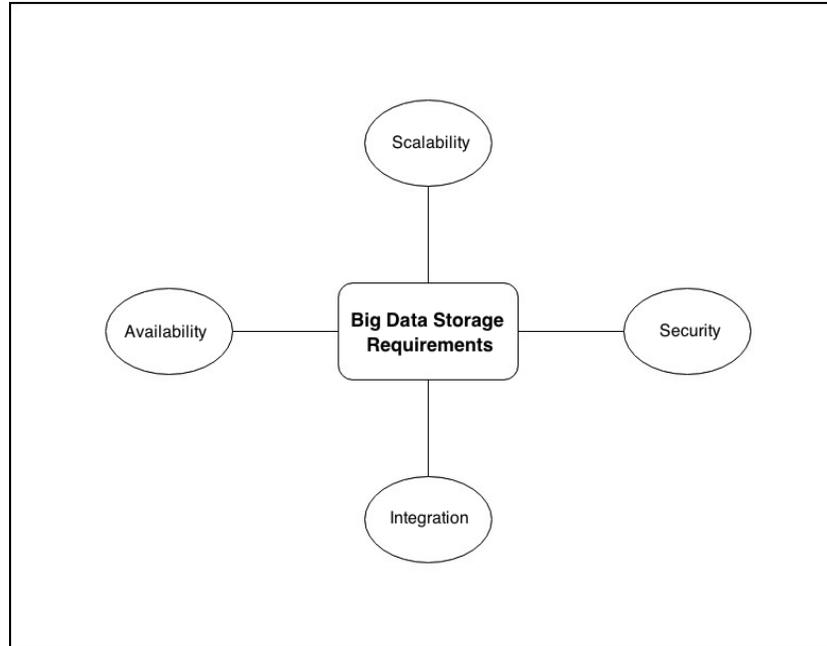


Fig. 1. Storage Requirements of a Big Data System

Technologies or solutions must be chosen on the basis of the specific requirements of a business or application. The available big data technologies offer different degrees of performance, security, and data capacity and integration capabilities [14]. Therefore, if the requirements are clear and precise, choosing a solution or combination of solutions to suffice the needs should not be difficult. This research paper performs a comparative study of 80 NoSQL solutions available for use in big data systems and elaborates on how requirements must be analyzed to determine the best solution for a concerned application.

Understandably, sustainability of heterogeneous environments requires data sharing between multiple processes, which can be performed using two possible ways. The first method makes use of APIs provided by independent solutions to retrieve data from them. For instance, if Drill [15] and HBase [4] are used together, Drill shall use the HBase API for data reads. Although, data sharing can be performed in real-time using this approach, it suffers from some potential issues [16].

The overheads for deserialization and serialization are very high and may cause serious bottleneck issues for applications like workflows. Moreover, there is no standardization in how the data should be represented. Developers perform custom integrations using their system and other systems. In order to manage these issues, another method for data sharing may be used, which enables sharing by means of common file formats that are optimized for high performance in analytical systems [17]. Owing to this, big data file formats form an important facet of big data storage optimization and are discussed in the later parts of this paper. This research work classifies data file formats and compares the advantages and disadvantages of each to provide a comprehensive comparison for use-case matching.

Recent surveys on NoSQL technologies [18, 19, 20] and data file formats [21, 22] discuss these two storage facets in isolation, which cannot be done in consideration of the fact that big data storage and its optimization need to be dealt with, in entirety, in this age of heterogeneous systems development that integrate different solutions to get the desired functional and performance results. Moreover, the coverage of this study is largest among available research papers. It covers 80 NoSQL solutions and reviews big data file formats comprehensively, which is not present in available literature. The rest of the research paper has been organized in the manner discussed below. Section 2 describes NoSQL and its characteristics. It aims to answer as to how NoSQL solves the multiple issues presented by the big data problem to traditional systems.

Big data storage models have been discussed in Section 3. The research paper provides a classification of

popular NoSQL solutions on the basis of data model-based categorization. The different categories are compared to provide an analysis of matching use-cases, to facilitate decision making in this domain. Section 4 discusses big data file formats in detail. All the available formats are divided into five categories and are compared to provide the pros and cons of using each of them. Moreover, best-suited applications for each of these formats are also discussed. Most of the available solutions are cloud-based, but big data storage is still not decentralized enough to provide optimum results. Future trends in big data technology deals with decentralized storage and blockchain. These prospects have been discussed in Section 5. Lastly, the research paper concludes and summarizes the findings in Section 6, providing insights for future work in this field.

## 2.0 NoSQL – A Solution for Big Data Storage Issues

The idea behind the development of relational databases was to provide a data storage approach that makes use of structured query language or SQL [23]. The introduction of these databases dates back to the 1970s when data schemas were not as complicated as they are today. Moreover, storage was expensive and not all data was considered for archival. With the rise in social media platforms, the amount of data being stored about events, objects and people has risen exponentially. The use of data in this time and age is not just limited to data archival, but it also extends to frequent data retrieval and processing, in order to serve purposes like generation of real-time feeds [24] and customized advertisements [25], in addition to many others.

Owing to the complexity of information being processed and the need to treat multiple, of the order of hundreds, database requests just to answer a single API request or render a webpage, the demands from modern database systems are ever-increasing. Some of the key drivers in this domain are the need for interactivity, increasing complexity and ever-evolving networks of users [26]. In order to serve these growing demands, sophisticated deployment strategies and improved computing infrastructure [188] are being put to use. With that said, single server deployments are expensive and highly complex, which has caused a drift towards the use of cloud hardware [27] for this purpose. Besides this, the use of agile methods has also reduced the development and deployment time [28], allowing quicker response to user needs.

It would not be wrong to state that relational databases were not created to manage the agility and scalability requirements of modern-day systems. Moreover, they are also not equipped to work with the cloud and take optimum advantage of its cheaper storage and processing capabilities. These shortcomings can be addressed using two main technical approaches, which have been discussed below –

- Manual Sharding

In order to make use of the distributed paradigm, tables need to be segmented into smaller units, which must then be stored across different machines. This process of splitting is called manual sharding [29]. However, this functionality is not available in a traditional database and needs to be implemented by the developer. Moreover, the storage of data on each instance is performed in an anonymous mode.

It is the responsibility of the application code to segment data, store it in a distributed manner, and perform query management and aggregate results to be presented to the user. Additional code shall be required for supporting data rebalancing, performing join operations, handling of resource failures and replication. It is crucial to mention that manual sharding may downgrade some of the benefits of relational databases like transactional integrity.

- Distributed Cache

Caching [30] is a commonly used process, which is primarily employed for improving the read performance of a system. It is noteworthy that the use of a cache has no impact on the write performance and is capable of adding substantially to the complexity of the overall system. Therefore, if the requirements from the system are read-intensive, then the use of distributed cache must be considered. On the other hand, write-intensive or read/write intensive applications do not require a distributed cache [31].

NoSQL databases [13] are known to mitigate the challenges associated with traditional databases. In addition, they also unleash the true power of cloud by making use of commodity hardware, which reduces the cost, and simplifies deployment, making the life of a developer much easier as there is no need to maintain multiple layers of cache

anymore.

Some of the advantages of NoSQL solutions over traditional databases are as follows –

- **Scalability**  
NoSQL allows systems to scale out horizontally [13]. Moreover, this can be done quickly without affecting the overall performance of the system with the help of cloud technologies. Scaling traditional databases require manual sharding that involves high costs and complexity. On the other hand, NoSQL solutions offer automatic sharding, reducing complexity as well as cost of the system [13].
- **Performance**  
As mentioned previously, NoSQL systems can be scaled out as required. With the increase in the number of systems, the performance of a system is also correspondingly improved. The fact that these systems involve automatic sharding means that the overhead associated with the same is also eliminated, which further contributes to the improved performance of the system.
- **High and Global Availability**  
Relational databases depend on primary and secondary nodes to fulfill the availability requirements. This not only adds to the complexity of the system, but it also makes the system moderately available. On the contrary, NoSQL solutions make use of master-less architecture and data is distributed across multiple systems [13]. Therefore, even upon the failure of a node, the availability of the application remains unaffected for read as well as write operations.  
NoSQL solutions offer data replication across resources [13]. Consequently, user experience is consistent irrespective of the location of the user. Moreover, it also plays a significant role in reducing latency with the added advantage of shifting the developer's focus from database administration to business primacies.
- **Flexible Data Modeling**  
It is possible to implement fluid and flexible data models in NoSQL [13]. This allows developers to implement query options and data types that benefit the application instead of those that suit the schema. In the process, the interaction between database and application is simplified, making this approach a better option for agile development.

NoSQL is an umbrella term used to describe a plethora of technologies, all of which entail the following common characteristics [32] –

### **2.1 Dynamic Schemas**

Relational databases [33] have an inherent requirement to create schemas in advance. Data is added to the database only after this requirement is fulfilled. For instance, if a system needs to store employee data like name, department, age, gender and salary, then the table created for the same must have the corresponding schema.

Such a requirement is unfit for agile development environments, as the fields of data might need to be changed over time. A new requirement may be added, as part of iteration, and subsequently, the schema may have to be altered. This is a time-consuming task if the database is large. As a result, the database may have to be shut for any use for a considerable amount of time to make required changes. Moreover, if the development process requires several iterations, the database may have to be shut rather frequently for significant amounts of time. Evidently, relational databases are inappropriate for storing data that are large, unstructured and unknown [33].

NoSQL satisfies this requirement since it has no predefined schemas. Moreover, data insertion does not require the developer to define a schema well in advance. As a result, changes to the data structure and data can be made in real-time without the need to shut the database for any other use [32]. There are several advantages of using this approach. Apart from the fact that it reduces administrator time, such an approach also reduces the time required for development and simplifies the process of code integration.

### **2.2 Auto-Sharding**

Relational databases are structured in such a manner that they need to have a server that controls the rest of the systems to provide reliability and availability requirements of a database solution. Therefore, such a system can only support vertical scaling [32], which is not just expensive, but it leads to creation of small number of points of failure. Besides this, it also places a limit on the amount of scaling that a system can support.

In view of the system requirements, a database solution must support horizontal scaling [13]. Therefore, it must be possible to add servers to the ensemble and get rid of the limitation that focuses on testing the capacity of a single server. Cloud Computing offers the best solution in this regard by providing on-demand services and unlimited scaling capacity [34]. So, the system no longer needs to rely on one server to fulfill its needs. Another important facet of using the Cloud is its inbuilt database administration. Moreover, the developer no longer needs to create complex platforms and can simply focus on writing the application code. Lastly, the use of Cloud-based, multiple servers cost significantly lesser than a high-capacity, single server.

In order to perform sharding of a database spanning across multiple servers, complex arrangements to make multiple servers act a single system, need to be put in place. On the other hand, NoSQL databases support auto-sharding. In other words, the database automatically distributes data across multiple systems without the need for the administrator to be aware of the server pool composition. Load balancing [35] for data and query are also automatically performed by the system. This allows the system to offer high availability. As and when the server goes down, it can be conveniently replaced and operations remain unaffected.

### **2.3 Automatic Replication**

Replication is performed automatically for any NoSQL system [32]. Therefore, the system can recover from disasters rather easily, also allowing high degrees of availability. From the developer's point of view, he or she no longer needs to cater for these facets of development in the application code.

### **2.4 Integrated Caching**

The integrated caching abilities [32] of NoSQL systems are rather well equipped and most of the frequently used data is kept in the system memory to ensure quick access. Therefore, there is no need to maintain multiple caching layers at the application level.

## **3.0 Big Data Storage Models**

NoSQL is a technology that is developed to counter the issues presented by relational databases, which is implemented in multiple ways by different models. Common characteristics of NoSQL models include efficient storage, reduced operational costs, high availability, high concurrency, minimal management, high scalability and low latency [36]. NoSQL solutions have been classified using multiple criteria.

Yen [37] provided a detailed classification of NoSQL solutions dividing them into nine categories, which include Wide Columnar Store, Document Store, Object Database, Tuple Store, Data Structures Server, Key Value Store, Key-Value Cache, Ordered Key Value Store and Eventually Consistent Key Value Store. Another taxonomical study was performed by North [37], which gave a comprehensive classification and included cloud-based solutions as well for the analysis. Solutions have been classified under six categories namely Entity-Attribute-Value Data Stores, Amazon Platform Column Stores, Key-Value Data Stores and Distributed Hash Table Document Stores.

Catel [39] and Leavitt [40] proposed a data model-based classification. Catel [39] divides NoSQL solutions into three categories namely Key Value Stores, Document Stores and Extensible Record Stores. On the other hand, Leavitt [40] proposes the use of three categories namely document-based, key value stores, column-oriented stores. Scofield [41] gave the most accepted categorization scheme by classifying databases into relational, graph, document, column and key value stores. This research paper uses the above-mentioned basis for classification and covers document-oriented stores, graph data model, key value store and wide column store in the following sections.

### **3.1 Document-Oriented Data Model**

A NoSQL database that uses documents for storage and retrieval of data is called document-oriented database [42]. Other names of this NoSQL data model are document database [43] and document store [44]. It is primarily used for management of semi-structured data because of its flexibility and support for variable schema. As mentioned previously, document databases use documents for its working. These documents may be in PDF or word format. However, blocks of JSON and XML are the more commonly used document formats. A relational database contains columns, which are described by their names and data types. On the contrary, in case of document databases, data type description and value for the concerned description are provided in a document [42]. The

structure of different documents making up a database may be similar or different in structure. Evidently, there is no need to alter the schema for adding data to the database.

Documents are grouped together to form a structure called collection [42]. There may be multiple collections in a database. This structure is similar in functioning to the table, which is present in a relational database [33]. Document-oriented databases provide a mechanism to execute queries on collections and retrieve documents that satisfy the attribute requirements. There are several advantages of using this approach, which include –

- Most of the growing data comes from IoT devices, social media and Internet. However, this data does not fit into standard application data models. Document-oriented databases offer flexible data modeling [42], in contrast to relational databases that force applications to fit data into existing models irrespective of their needs.
- The write performance of document-oriented databases is better than conventional systems [45]. In order to make a system available for writing, the database can compromise on data consistency as well. Therefore, even if a system fails and replication takes longer than expected, the write operation will be fast.
- The indexing features and query engines of databases available in this category are known to be fast and efficient [42]. Therefore, they offer faster query performance.

### **3.2 Graph Data Model**

This NoSQL data model is tailor made to support storage and processing of voluminous data, which may be semi-structured, structured or unstructured, in type. Therefore, data can be accessed and acquired from different sources. As a result, Graph data model [46] is popularly used in social media [47] and big data analytics [48]. It is noteworthy that relational databases were developed for storing structured information available in and generated by enterprises. Therefore, schema of the data to be stored is available beforehand. On the contrary, data generated by IoT (Internet of Things) and social media is unstructured. Moreover, it is being generated in real time.

Graph databases [49] are a good option for storing unstructured data generated by such diverse sources at high velocity. There is no need to define a schema before storing data, which makes the database rather flexible. Besides this, graph databases are cost-effective and dynamic when it comes to integration of data coming from different sources [49]. Moreover, graph databases are better equipped to handle, store and process high-velocity data as compared to relational databases.

Aforementioned applications like social media analytics and IoT-based analytical solutions [187] require the base technology to integrate data coming from heterogeneous sources and establish links between the different datasets created. Application data of this kind can best be handled using a semantic graph database or RDF triplestore [50], which focuses on relationships between different elements of the database and generate analytics on this basis. These graph databases are primarily used for real time analytics because of their ability to handle large datasets, without the need to define a schema in advance.

The benefits of using semantic graph database can be summarized as follows –

1. Integration of inbound data from different sources is limited when the schema needs to be defined before adding data because the addition of a new source might require a change in schema, which is both time-consuming as well as complicated. In databases where there is no such need, data integration is limitless, simple and cost-effective [51].
2. Semantic graph databases offer an additional support to ontologies or semantically rich data schemas [51]. Therefore, organizations can create logical models in any way they desire.
3. Semantic graph databases use international standards for data representation on the web [51]. This results in easier integration and sharing of data. Uniform Resource Identifier (URI) [52] is one of the standards used for data representation in semantic graph databases. URI is a unique ID, which is used to distinguish between linked entities. The presence of such a clear approach for entity identification makes access and search easier, making the approach cost-effective. Moreover, it also makes data sharing easier as far as mapping data to Linked (Open) Data is concerned. In addition, challenges like vendor lock-in can be avoided.

### **3.3 Key-Value Data Model**

The most flexible type of NoSQL database is Key-Value Store [53], which implements schema-less policy by having no schema and making the data value opaque. The data value can store strings, numbers, images, binaries, counters, XML, JSON, HTML and videos, in addition to many others [53]. The stored values can be accessed with the help of a key. The flexibility of the database is manifested in the fact that the application controls the data value, completely. Key benefits of key-value stores are as follows –

1. The database does not force the application to structure its data in a specific form. Therefore, the application is free to model its data in accordance with the requirements of the use case.
2. Objects can simply be accessed with the help of the key assigned to the object. When using this database, there is no need to perform operations like union, join and lock on objects [53], which make this data model, most efficient and high performing.
3. Most of the available key-value databases allow scale out as and when the demand for the same arises. Moreover, this can be done using commodity hardware without the need for any redesigning.
4. Providing high availability is much easier and uncomplicated with key-value stores. The distributed architecture and master-less configuration of some of the available databases of this type ensures higher resilience [53].
5. The design of these databases is such that it is simple to add and remove capacity. Moreover, these databases are better equipped to deal with network failures and hardware malfunctions [53], lowering the downtime considerably.

### **3.4 Wide-Column Data Model**

Wide Column Stores [54] have columns and column families, as base entities. Facts or data are grouped together to form columns, which are further organized in the form of column families that are constructs similar to tables in relational databases. For example, data about an individual like name, account name and address are facts about the individual and can be grouped together to form a row in a relational database. On the contrary, same facts are organized in the form of columns in a wide-column store and each of the columns includes multiple groups. Therefore, a single wide-column can store data equivalent to the same stored by many rows in a relational database. Other names of such databases include column-oriented DBMS [55], columnar databases [56] and column families [57].

Key advantages [54] of using wide column store databases include –

1. Partitioning and data compression can be performed efficiently using wide column store databases.
2. Aggression queries like AVG, SUM and COUNT can be performed effectively and efficiently because of the inherent structure of this database.
3. This database type is highly scalable and well suited for massively parallel processing (MPP) systems.
4. Tables with huge amounts of data can be loaded and queried in almost no time, making the response time of the database relatively low.

### **3.5 Choosing a NoSQL Solution for a Big Data System**

One of the major technological decisions to be made while designing a big data application include selecting a NoSQL solution. For this, two things need to be specifically kept in mind. The right data model for an application depends on the type of data that needs to be dealt with. A classification of NoSQL databases on the basis of their data model is provided in the previous section. The success of a big data application can be greatly impacted by a mismatch in the data model of the application and that of the chosen NoSQL solution. Another important consideration in choosing a NoSQL solution is the scalability requirement. It is critical to understand that there are some NoSQL solutions that can scale well like Cassandra whereas others may be memory-based and fail to scale across machines.

As mentioned previously, the right data model for an application depends on the data that it is expected to deal with. For instance, if the application's data can be represented in the form of a graph, then the graph model is most

appropriate data model for the application. Each data model best suits to a specific set of applications and requirements. This section discusses the selection criteria for deciding which big data model befits a particular case study in view of its domain model, data access patterns and use cases.

Document databases work around documents. Therefore, a document is the basic atomic unit of storage in such databases. Any domain model that allows splitting and partitioning of its data across documents can use a document database. Some common case studies include CMS, blog-software and wiki-software [58]. However, when considering this data model, you may come across use-cases where a relational model may be just as good an option to use as the non-relational database.

Key-value stores are commonly used data models, preferred for application areas surrounding data like user profile, emails, blog/article comments, session information, shopping cart data, product reviews, product details and Internet Protocol (IP) forwarding tables [59], in addition to many others. It is crucial to understand that a key-value store can be used to store complete webpages [60]. In this case, URL can be used as the key, and webpage content, as value. However, other data models may be better suited for this purpose if the application requires so.

Graph databases offer an effective way to manage and combine data. Enterprise data is typically linked and graph databases for their storage ensure easier management of content. Moreover, personalization can also be achieved in a simpler manner. In addition to this, the concept of connected world, which has particularly picked up pace after the rise of social media and IoT, can take advantage of the fact that graph databases allow integration of heterogeneous, interlinked data from different sources.

Wide-column stores form the last category of big data models. These databases are deemed most appropriate for distributed systems [61]. In other words, if the data available is large and can be split across machines, then a wide-column store database can be extremely useful. Some of the primary advantages of using this database is reduced query time for some queries. However, this point must be clearly investigated before a decision in favor of such a solution is taken. For some queries, the time may be same or higher than that offered by conventional RDBMS solutions [62]. The use-cases for the four big data models have been summarized in Table 1.

Table 1: Use Cases for Different Big Data Models

| Big Data Model    | Preferred for Use Cases  | Not Preferred for Use Cases  |
|-------------------|--|--|
| Document-Oriented | <ol style="list-style-type: none"> <li>1. Content management systems</li> <li>2. E-commerce platforms</li> <li>3. Blogging platforms</li> <li>4. Analytics platforms</li> </ol>  | <ol style="list-style-type: none"> <li>1. Applications requiring complex search queries.</li> <li>2. Applications requiring complex transactions with multiple operations.</li> </ol>  |
| Key-Value         | <ol style="list-style-type: none"> <li>1. Storage of user preferences.</li> <li>2. Maintenance of user profiles that don't have a specific schema.</li> <li>3. Storage of session data for users.</li> <li>4. Storage of shopping carts' data for multiple users.</li> </ol> | <p>In scenarios where –</p> <ol style="list-style-type: none"> <li>1. Specific data value needs to be queried.</li> <li>2. Multiple unique keys need to be worked upon.</li> <li>3. Frequent update of a part of the value.</li> <li>4. Data values have established relationships with each other and the application requires exploitation of the same.</li> </ol> |
| Graph             | <ol style="list-style-type: none"> <li>1. Network and IT operations</li> <li>2. Graph based searches</li> <li>3. Social networks</li> <li>4. Fraud detection</li> </ol>  | Such a model is inappropriate for any application for which the data cannot be modeled as a graph.   |
| Wide-Column       | <ol style="list-style-type: none"> <li>1. Blogging platforms</li> <li>2. Content management systems</li> <li>3. Counter-based systems</li> <li>4. Applications with write-intensive processing</li> </ol>  | <ol style="list-style-type: none"> <li>1. Application requires complex querying.</li> <li>2. Application has varying patterns of queries.</li> <li>3. In scenarios where the database requirement is not established, the use of such a store must be avoided.</li> </ol>  |

While discussing the applicability of NoSQL solutions to real-world problem, it is important to mention CAP Theorem [63]. This theorem introduces the concept of Partition Tolerance (P), Availability (A) and Consistency (C) for distributed systems and states that all these three characteristics cannot be ensured by a solution simultaneously. Consistency is a characteristic that ensures that all the nodes of the distributed system must read the same value of

data at all times. If a change in data value is made, then the change must be consistent for all nodes. However, if the change results in an error, then a rollback must be performed to ensure consistency.

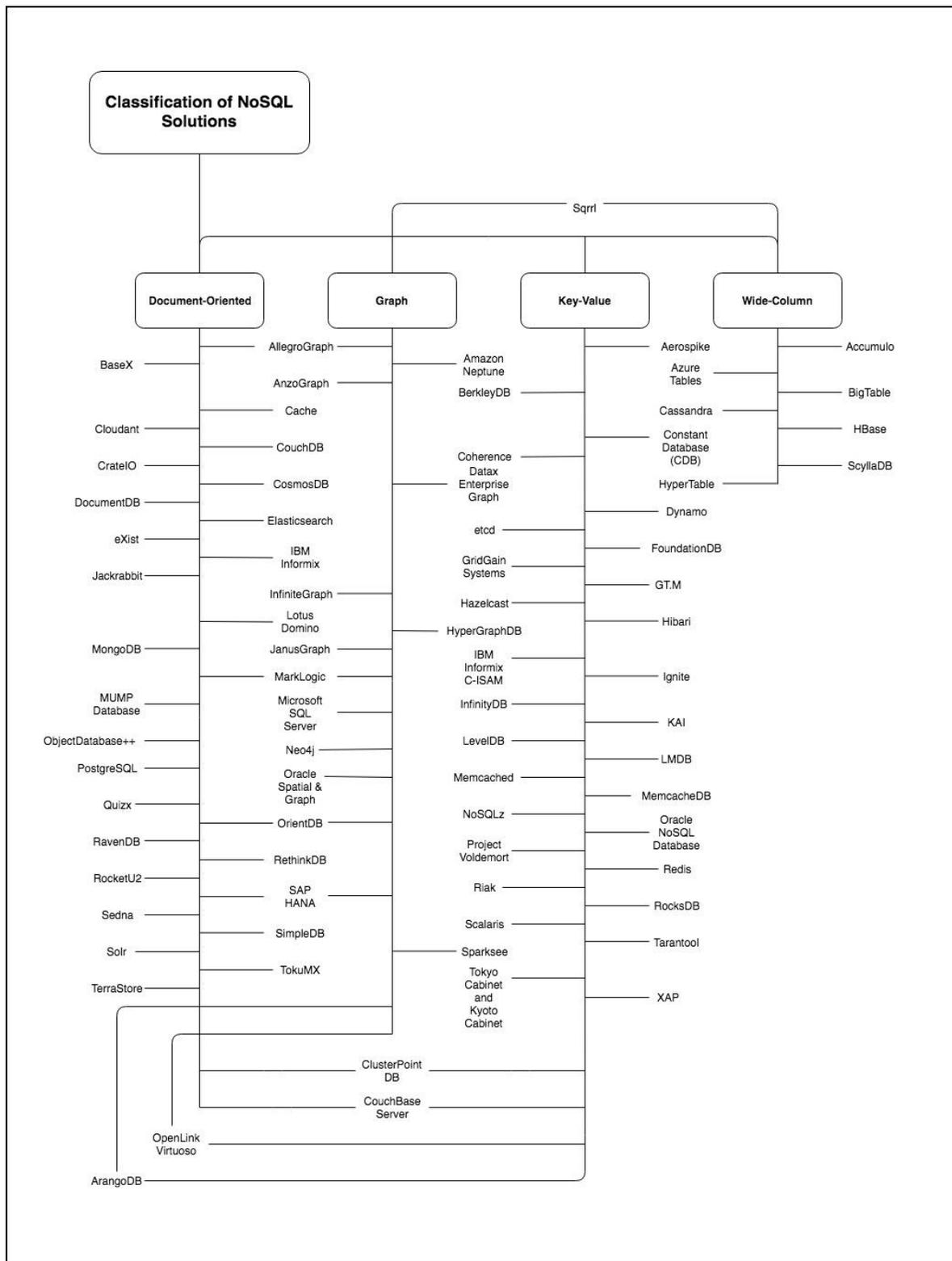


Fig. 2. Data Model-Based Classification of NoSQL Solutions

Availability defines the operational requirement of the system that ensures that as and when a request is made to the system by a user, it must be respond to it despite its state. Partition tolerance refers to a system’s ability to operate despite failure of a partition and message loss. It can also be described as the ability of a system to operate irrespective of network failure. Different database solutions and their CAP status has been described in the following sections. It has been stated that no distributed system can possess all the three characteristics. On the basis of this assertion, NoSQL systems can be CA (Consistent-Available), AP (Available-Partition Tolerant) or CP(Consistent-Partition Tolerant) [64].

One of the biggest challenges in the use of NoSQL is the use of an appropriate data model. The use of an inappropriate model can affect system performance significantly, which makes this a crucial technological decision. Another important technological decision in this case is choosing the right distribution model. Scaling of read operations is supported by master-slave architecture. However, if scaling of both read and write operations is desired, then peer-to-peer architecture is a better option. The use of NoSQL databases also has some security issues that must be considered and mitigated before a solution can be developed and deployed using the same. Fig. 2 illustrates data model-based classification of NoSQL solutions. A detailed description of implementations along with the supported data models is provided in Table 2. The letter Y indicates support for the corresponding data model while N suggests no support for the concerned data model.

Table 2: Comparison of Solutions Based on Different Data Models

| S. No. | Implementation      | Supported Data Model |       |           |             | Key Features  |
|--------|---------------------|----------------------|-------|-----------|-------------|---|
|        |                     | Document-Oriented    | Graph | Key-Value | Wide-Column |   |
| 1.     | AllegroGraph [65]   | Y                    | Y     | N         | N           | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Supports RDF, JSON and JSON-LD</li> <li>3. Provides Multi-Master Replication</li> <li>4. Supports ACID Transactions, two-phase commit and full-text search</li> </ol>   |
| 2.     | Accumulo [66]       | N                    | N     | N         | Y           | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Scalable and distributed</li> <li>3. It is built over and above Hadoop [8], Thrift [74] and Zookeeper [107]</li> <li>4. Provides cell-level security and mechanisms for server-side programming</li> </ol>                           |
| 3.     | Aerospike [67]      | N                    | N     | Y         | N           | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Highly scalable</li> <li>3. Flash-optimized, in-memory</li> <li>4. Reliable and consistent</li> <li>5. Used for applications like dynamic web portals, user profiling and fraud detection</li> </ol>                                 |
| 4.     | Amazon Neptune [68] | N                    | Y     | N         | N           | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Fully managed database</li> <li>3. Provided as a web service</li> <li>4. Supports RDF and property graph models</li> <li>5. Supports SPARQL [196] and TinkerPop Gremlin [197] query languages</li> </ol>                      |
| 5.     | AnzoGraph [69]      | N                    | Y     | N         | N           | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Massively parallel</li> <li>3. Graph Online Analytics Processing (GOLAP) database</li> <li>4. Supports SPARQL [196] and Cypher [145]</li> <li>5. Originally designed to analyze semantic triple data interactively</li> </ol> |
| 6.     | ArangoDB [70]       | Y                    | Y     | Y         | N           | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Supports multiple database models with a single core</li> <li>3. Possesses unified query language called ArangoDB Query Language (AQL)</li> </ol>  |

|     |                               |   |   |   |   |   |
|-----|-------------------------------|---|---|---|---|---|
| 7.  | Azure Tables [71]             | N | N | N | Y | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Provisioned as a service where storage of data is allowed into collections that can be partitioned. Data is accessed by means of primary and partition keys.</li> </ol>   |
| 8.  | BaseX [72]                    | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Provides support for JSON, XML and binary formats</li> <li>3. Implements master-slave architecture</li> <li>4. Provides support for concurrent structural and full-text update/search</li> </ol>   |
| 9.  | BerkeleyDB [73]               | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free (with commercial versions also available)</li> <li>2. High performing and scalable</li> <li>3. Supports complex data management</li> <li>4. Most appropriate for applications requiring embeddable database.</li> </ol>  |
| 10. | BigTable [75]                 | N | N | N | Y | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. High performance</li> <li>3. Provides data compression</li> </ol>   |
| 11. | Cache [76]                    | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Data is stored in multi-dimensional arrays. Therefore, structured data that is hierarchical in nature can be stored.</li> <li>3. Commonly used for business and health-related applications</li> </ol>  |
| 12. | Cassandra [77]                | N | N | N | Y | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed</li> <li>3. Highly available</li> <li>4. Master-less replication with robust support for clusters across multiple datacenters</li> </ol>   |
| 13. | CDB or Constant Database [78] | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free library</li> <li>2. On-disk associative array that maps keys to values, allowing a key to have multiple values</li> <li>3. It can be used as a shared library.</li> </ol>  |
| 14. | Cloudant [79]                 | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Distributed database service</li> <li>3. Uses BigCouch [189] and JSON model, at backend</li> </ol>  |
| 15. | Clusterpoint Database [80]    | Y | N | Y | N | <ol style="list-style-type: none"> <li>1. Proprietary, but allows download for free</li> <li>2. Distributed JSON/XML database platform</li> <li>3. Transactions are compliant with ACID properties</li> <li>4. Highly available</li> <li>5. Provides sharding and replication</li> <li>6. Uses SQL or JS as query language</li> </ol> |
| 16. | Coherence [81]                | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. In-memory data grid and distributed cache</li> <li>3. Appropriate for systems requiring high scalability and availability keeping latency at lower levels</li> </ol>  |
| 17. | CouchBase Server [82]         | Y | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed database</li> <li>3. Uses SQL as querying language</li> <li>4. Uses JSON model</li> </ol>  |
| 18. | CouchDB [83]                  | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Supports JSON over HTTP/REST</li> <li>3. Provides limited support for ACID transactions</li> <li>4. Supports multi-version concurrency control</li> </ol>  |
| 19. | CrateIO [84]                  | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Free</li> </ol>   |

|     |                                |   |   |   |   |   |
|-----|--------------------------------|---|---|---|---|---|
|     |                                |   |   |   |   | <ul style="list-style-type: none"> <li>2. It is based on Elasticsearch/Lucene ecosystem</li> <li>3. Supports objects that are binary or are also called BLOBs.</li> <li>4. Makes use of SQL syntax for distributed querying of the system in real time</li> </ul>                         |
| 20. | CosmosDB [85]                  | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Provisioned as Platform-as-a-Service</li> <li>3. Based on DocumentDB</li> </ul>   |
| 21. | DataStax Enterprise Graph [86] | N | Y | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Scalable, distributed database</li> <li>3. Allows real-time querying</li> <li>4. Supports Tinkerpop [197]</li> <li>5. It is known to integrate well with Cassandra [77]</li> </ul>                                      |
| 22. | DocumentDB [87]                | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Provisioned as a database service</li> <li>3. Fully managed version of MongoDB</li> </ul>   |
| 23. | Dynamo [88]                    | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Distributed datastore</li> <li>3. Highly available</li> <li>4. Supports incremental scalability, symmetry among nodes, decentralization and it exploits the heterogeneity of the infrastructure it works on.</li> </ul> |
| 24. | ElasticSearch [89]             | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Supports JSON</li> <li>3. Basically a search engine</li> </ul>   |
| 25. | etcd [90]                      | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Supports binary data</li> <li>3. Allows versioning, validation, collections, triggers, clustering, Lucene full text search, ACLS and XQuery Update</li> <li>4. Uses XML over REST/HTTP</li> </ul>                              |
| 26. | eXist [91]                     | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Supports text, JSON, HTML and XML formats, in addition to binary formats</li> <li>3. XQuery is the provided querying language while XSLT is the corresponding programming language</li> </ul>                                  |
| 27. | FoundationDB [92]              | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Complies with ACID properties</li> <li>3. Scalable</li> <li>4. Allows replications</li> <li>5. Bindings for Python, C, PHP and Java, in addition to many other programming languages is available</li> </ul>                   |
| 28. | GridGain Systems [93]          | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Services and software solutions are provided for systems dealing with big data</li> <li>3. Supports in-memory computing</li> <li>4. Provides improved throughput and reduced latency</li> </ul>                         |
| 29. | GT.M [54]                      | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Developed for transaction processing</li> <li>3. Supports ACID transactions</li> <li>4. Supports replication and database encryption</li> </ul>  |
| 30. | Hazelcast [95]                 | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. MapStore can be defined by the user</li> <li>3. MapStore can be persistent</li> <li>4. High consistency and supports sharing in the form of consistent hashing</li> </ul>  |

|     |                           |   |   |   |   |   |
|-----|---------------------------|---|---|---|---|---|
| 31. | HBase [4]                 | N | N | N | Y | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed database</li> <li>3. It runs on top of Hadoop and provides capabilities similar to that of BigTable.</li> <li>4. It is fault-tolerant for scenarios where a large amount of sparse data is being dealt with.</li> </ol>  |
| 32. | Hibari [96]               | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed big data store</li> <li>3. Highly available</li> <li>4. Strongly consistent</li> </ol>   |
| 33. | HyperGraphDB [97]         | N | Y | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Schemas are dynamic and flexible</li> <li>3. Knowledge representation and data modeling are efficient</li> <li>4. Non-blocking concurrency</li> <li>5. Appropriate for semantic web and arbitrary graph use cases</li> </ol>   |
| 34. | HyperTable [98]           | N | N | N | Y | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. It is based on BigTable</li> <li>3. Massively scalable</li> </ol>   |
| 35. | IBM Informix [99]         | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. RDBMS that supports JSON</li> <li>3. Complies with ACID rules</li> <li>4. Supports sharding and replication</li> </ol>  |
| 36. | IBM Informix C-ISAM [100] | N | N | Y | N | <ol style="list-style-type: none"> <li>1. This API complies with Open Standards</li> <li>2. Allows management of data files, which have been organized using B+ indexing</li> <li>3. It is the file storage used by Informix [99].</li> </ol>   |
| 37. | Ignite [101]              | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed, in-memory computing platform</li> <li>3. Provides caching and processing platform</li> <li>4. Provides support for ACID transactions and MapReduce jobs</li> <li>5. Allows partitioning, clustering and replication</li> <li>6. Highly consistent</li> </ol>  |
| 38. | InfiniteGraph [102]       | N | Y | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Cloud-enabled</li> <li>3. Distributed</li> <li>4. It is scalable and cross-platform</li> <li>5. It is capable of handling high throughput</li> </ol>  |
| 39. | InfinityDB [103]          | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Completely developed in Java and includes DBMS and database engine</li> <li>3. Based on B-tree architecture</li> <li>3. Provides high performance</li> <li>4. Reduces risks associated with failures</li> </ol>   |
| 40. | Jackrabbit [104]          | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Implementation of Java Content Repository</li> </ol>   |
| 41. | JanusGraph [105]          | N | Y | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed</li> <li>3. Scalable and integrates well with backend databases like HBase [4], Cassandra [77], BigTable [75] and BerkleyDB [73]</li> <li>4. Integrates well with platforms like Giraph [144], Spark [8] and Hadoop [12]</li> <li>5. Provides support for full text search by external integration with Solr [137] and Elasticsearch [89]</li> </ol> |
| 42. | KAI [106]                 | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Scalable</li> </ol>  |

|     |   |   |   |   |   |   |
|-----|---|---|---|---|---|---|
|     |   |   |   |   |   | <ul style="list-style-type: none"> <li>3. Highly fault-tolerant</li> <li>4. Provides low latency</li> <li>5. Used for social networks and web repositories</li> </ul>   |
| 43. | LevelDB [108]                                 | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Maintains byte arrays for storing key and value pairs.</li> <li>3. Data compression is supported by means of Snappy</li> <li>4. Supports forward/backward iteration and batch writing</li> <li>5. Used as a library</li> </ul>               |
| 44. | Lightning Memory-Mapped Database (LMDB) [109] | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Embedded database</li> <li>3. High performance</li> <li>4. Provides API bindings for many programming languages.</li> <li>5. Employs multi-version concurrency control is offers high levels of reliability</li> </ul>                       |
| 45. | Lotus Domino [110]                            | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. It is a multi-value database [190]</li> </ul>   |
| 46. | Marklogic [111]                               | Y | Y | N | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Supports XML, JSON and RDF triples</li> <li>3. Distributed</li> <li>4. Provides high availability, full-text search, ACID compliance and security</li> </ul>   |
| 47. | Memcached [112]                               | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Memory caching system that is general purpose and distributed</li> <li>3. Scalable architecture</li> <li>4. Supports sharding</li> </ul>   |
| 48. | MemcacheDB [113]                              | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. A version of memcached that has persistence</li> <li>3. It is a memory caching system that is distributed and general purpose.</li> <li>4. Development has halted on this solution.</li> </ul>   |
| 49. | Microsoft SQL Server [114]                    | N | Y | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Typically used for modeling many-to-many relationships between data</li> <li>3. Integration of relationships are done into Transact-SQL and the foundation DBMS is SQL Server</li> </ul>  |
| 50. | MongoDB [6]                                   | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Supports BSON or binary JSON</li> <li>3. Allows replication and sharding</li> </ul>  |
| 51. | MUMP Database [115]                           | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. MUMPS is a programming language with inbuilt database</li> <li>3. Used for applications related to health sector</li> </ul>   |
| 52. | Neo4j [116]                                   | N | Y | N | N | <ul style="list-style-type: none"> <li>1. Free</li> <li>2. Can be used for ACID transactions</li> <li>3. Supports clustering and high availability</li> <li>4. Provides complete administrative support</li> <li>5. Provides inbuilt REST API for interface with other programming languages</li> </ul> |
| 53. | NoSQLz [117]                                  | N | N | Y | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Complies with ACID properties</li> <li>3. Allows CRUD (Create, Read, Update, Delete) operations</li> <li>4. Easy to implement</li> </ul>  |
| 54. | ObjectDatabase++ [118]                        | Y | N | N | N | <ul style="list-style-type: none"> <li>1. Proprietary</li> </ul>  |

|     |                                |   |   |   |   |   |
|-----|--------------------------------|---|---|---|---|---|
|     |                                |   |   |   |   | 2. Binary<br>3. Structure of native C++ class   |
| 55. | OpenLink Virtuoso [119]        | Y | Y | Y | N | 1. Proprietary<br>2. Hybrid of database engine and middleware<br>3. High performance and secure<br>4. Supports SQL [23] and SPARQL [196] for performing operations on SQL tables and RDF<br>5. JSON, XML and CSV document types are supported                             |
| 56. | Oracle NoSQL Database [120]    | N | N | Y | N | 1. Proprietary<br>2. Supports horizontal scalability and transparent load balancing<br>3. Supports replication and sharding<br>4. It is highly available and fault-tolerant   |
| 57. | Oracle Spatial and Graph [121] | N | Y | N | N | 1. Proprietary<br>2. Capable for handling RDF and property graphs   |
| 58. | OrientDB [122]                 | Y | Y | N | N | 1. Free<br>2. Supports JSON over HTTP<br>3. Supports SQL-type language use<br>4. Can be used for ACID transactions<br>5. Supports sharding, multi-master replication, security features and schema-less modes   |
| 59. | PostgreSQL [123]               | Y | N | N | N | 1. Free<br>2. Supports JSONB, JSON function and JSON store<br>3. Supports HStore 2 and HStore [192]   |
| 60. | Project Voldemort [124]        | N | N | Y | N | 1. Free<br>2. Supports horizontal scalability<br>3. Availability is high for read/write operations<br>4. Fault recovery is transparent<br>5. Supports automatic partitioning and replication<br>6. Considered appropriate for applications with read-intensive operations |
| 61. | Qizx [125]                     | Y | N | N | N | 1. Proprietary<br>2. Distributed XML database<br>3. Supports text, JSON and binaries<br>4. Provides integrated full text search   |
| 62. | RavenDB [126]                  | Y | N | N | N | 1. Free<br>2. Fully transactional and high performance<br>3. Highly available<br>4. Multi-platform and easy to use<br>5. Multi-model architecture that allows it to work well with SQL systems  |
| 63. | Redis [127]                    | N | N | Y | N | 1. Free<br>2. Read/write access is efficient<br>3. Fault-tolerant<br>4. Supports automatic partitioning<br>5. Appropriate for applications involving structured strings   |
| 64. | RethinkDB [128]                | Y | N | N | N | 1. Free<br>2. Distributed database<br>3. Supports JSON<br>4. Provides sharding and replication  |
| 65. | Riak [129]                     | N | N | Y | N | 1. Free<br>2. Highly available and fault-tolerant<br>3. Highly scalable and simple to operate<br>4. Cloud storage and enterprise versions of Riak are also available<br>5. It supports automatic data distribution and  |

|     |                 |   |   |   |   |  |
|-----|-----------------|---|---|---|---|--|
|     |                 |   |   |   |   | replication for resilience and improved performance.   |
| 66. | RocketU2 [130]  | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Provides dynamic support</li> <li>3. Scalable</li> <li>4. Reliable and efficient</li> <li>5. Appropriate for business information management</li> </ol>  |
| 67. | RocksDB [131]   | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Embedded database that assures high performance</li> <li>3. Supports all the features of LevelDB [108]. In addition, it also supports geospatial indexing, universal compaction, column families and transactions.</li> </ol>   |
| 68. | SAP HANA [132]  | Y | Y | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Supports JSON only</li> <li>3. Can be used for ACID transactions</li> </ol>  |
| 69. | Scalaris [133]  | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Highly available and fault-tolerant</li> <li>3. Massively scalable</li> <li>4. Consistent</li> <li>5. Self-managing</li> <li>6. Minimal maintenance overhead</li> <li>7. Considered appropriate for applications that are read/write intensive</li> </ol>                     |
| 70. | ScyllaDB [134]  | N | N | N | Y | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Distributed</li> <li>3. Designed for integration with Cassandra for reducing latency and improving throughput</li> <li>4. It supports Thrift and CQL, protocols also supported by Cassandra</li> </ol>  |
| 71. | Sedna [135]     | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. XML database</li> </ol>   |
| 72. | SimpleDB [136]  | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Distributed database</li> <li>3. Used as web service in concert with Amazon EC2 [193] and S3 [194]</li> <li>4. Provides availability and partition tolerance</li> </ol>  |
| 73. | Solr [137]      | Y | N | N | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Search engine written in Java</li> <li>3. Supports real-time indexing, full text search, database integration, dynamic clustering and rich document handling</li> <li>4. Provides index replication and distributed search</li> <li>5. Scalable and fault tolerant</li> </ol> |
| 74. | Sparksee [138]  | N | Y | N | N | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Scalable</li> <li>3. High performance</li> <li>4. First graph database for mobiles</li> <li>5. Bindings available for C++, C#, Objective C, Python and Java</li> </ol>   |
| 75. | Sqrl [139]      | N | Y | N | Y | <ol style="list-style-type: none"> <li>1. Proprietary</li> <li>2. Distributed</li> <li>3. Mass-scalable</li> <li>4. Real-time database</li> <li>5. Provides cell-level security</li> </ol>   |
| 76. | Tarantool [140] | N | N | Y | N | <ol style="list-style-type: none"> <li>1. Free</li> <li>2. Provides crash resistance with the help of</li> </ol>   |

|     |                                       |   |   |   |   |  |
|-----|---------------------------------------|---|---|---|---|--|
|     |                                       |   |   |   |   | maintenance of write ahead logs<br>3. Can be integrated with other applications and frameworks written in different programming languages.   |
| 77. | TokuMX [141]                          | Y | N | N | N | 1. Free<br>2. Version of MongoDB [6]<br>3. Supports fractal tree indexing [195]  |
| 78. | TerraStore [143]                      | Y | N | N | N | 1. Free<br>2. In-memory storage<br>3. Dynamic cluster configuration<br>4. Persistent<br>5. Supports load balancing and automatic data redistribution<br>6. Used for structured big data                                |
| 79. | Tokyo Cabinet and Kyoto Cabinet [142] | N | N | Y | N | 1. Free<br>2. Provides two libraries for database management<br>3. Storage is done via hash tables and B+ trees<br>4. Provides limited support for transactions  |
| 80. | XAP [146]                             | N | N | Y | N | 1. Proprietary<br>2. Software platform for in-memory computing<br>3. Appropriate use cases for this solution include real-time analytics and transaction processing requiring low latency and high performance levels. |

#### 4.0 Big Data File Formats: Storing Data in Hadoop Ecosystem

Hadoop [12] offers one of the most cost-effective and efficient ways to store data in huge amounts. Moreover, structured, semi-structured and unstructured data types can be stored and then, processed using tools like Pig [147], Hive [148] and Spark [8] to gain the results required for any future analysis or visualization. Hadoop allows the user to store data on its repository in several ways. Some of the commonly available and understandable working formats include XML [149], CSV [150] and JSON [151].

Although, JSON, XML and CSV are human-readable formats, but they are not the best way, to store data, on a Hadoop cluster. In fact, in some cases, storage of data in such raw formats may prove to be highly inefficient. Moreover, parallel storage is not possible for data stored in such formats. In view of the fact that storage efficiency and parallelism are the two leading advantages of using Hadoop, the use of raw file formats may just defeat the whole purpose.

CERN [152] had chosen four candidate technologies, ORC [153], Parquet [154], Kudu [155] and Avro [156] for this purpose. However, we have included other data file formats like Arrow [157] and text-based formats to this discussion for comprehensibility. Fig. 3 shows the classification and hierarchy of big data file formats. Each of these file formats has a set of advantages and disadvantages to its name. This section explores these facets of big data file formats and shall provide a discussion on the criteria that must be used for selecting a file format.

##### 4.1 Text-Based Formats

Simplest example of such a data file format is entries stored line-by-line terminated by a newline character or carriage return. In order to compress data, a file-level compression codec like BZIP2 [ ] needs to be used. Three formats namely text or CSV [150], JSON [151] and XML [149] are available under this category.

###### 4.1.1 Plain Text

Data stored in this format is mostly formatted in such a manner that fields either has fixed width or a delimiter separated entries, as is the case of CSV in which entries are separated using commas.

###### 4.1.2 XML

It is possible to use external schema definitions in XML. However, the performance of serialization and

deserialization is usually poor.

#### 4.1.3 JSON

JavaScript Object Notation (JSON) [158] is more performance effective than XML, but the problem with serialization and deserialization performance exists.

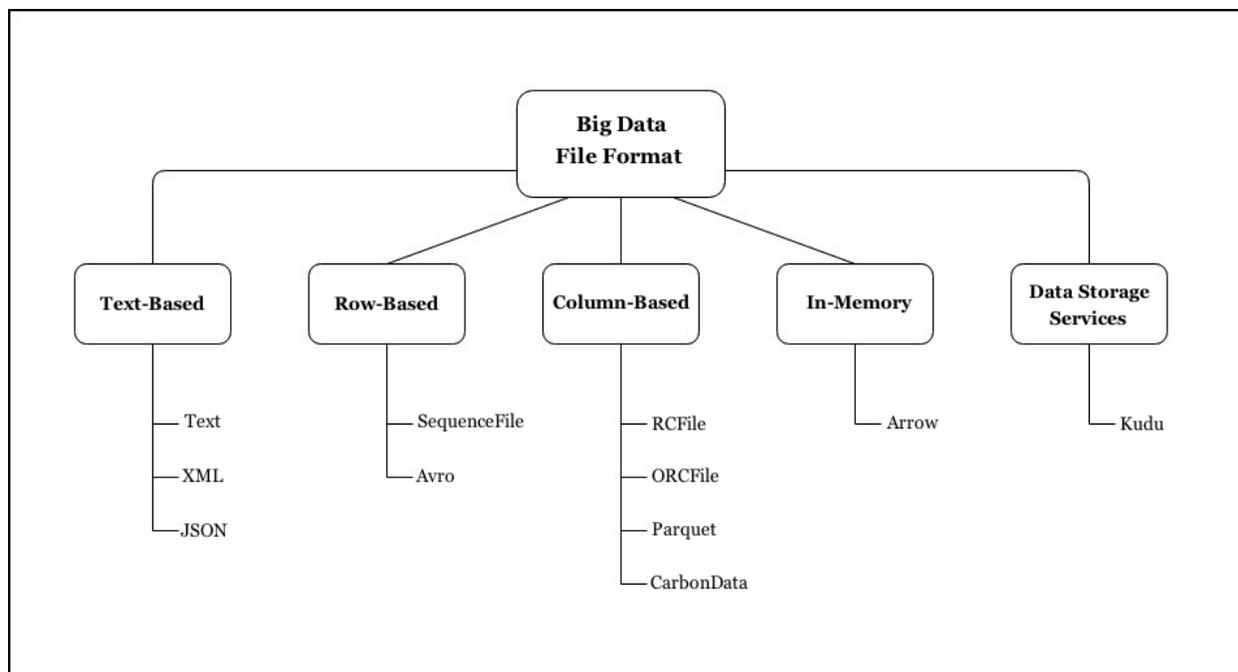


Fig. 3. Classification of Big Data File Formats

## 4.2 Row-Based Formats

### 4.2.1 SequenceFile

This file format is supported as part of the Hadoop framework as part of which a large file has container of binary key-value pairs for storing multiple files of small size. Therefore, the key-value pairs corresponding to records are encoded. The integration of SequenceFile [159] data file format is smooth with Hadoop in view of the fact that the former was developed for the latter.

### 4.2.2 Avro

Apache Avro [156] is used for compact binary format as a data serialization standard. Typically, it is used to store persistent data related to communication protocols and HDFS [5]. One of the major advantages of using Apache Avro is high ingestion performance, which is attributed to fast and lightweight deserialization and serialization, provided by the same. It is important to mention that Avro does not have an internal index. However, the directory-based partitioning technique available in HDFS can be used for facilitating random access of data. Data compression algorithms supported by Apache Avro include DEFLATE [160] and Snappy [161].

There are other serialization and deserialization frameworks like Thrift and Protocol buffers [162] that stand in competition with Avro. However, Avro is a built-in component of Hadoop while these frameworks are external. Besides this, schema definition in Avro is done using JSON whereas Thrift and Protocol buffers depend on Interface Definition Languages (IDLs) [162] for defining the schema.

## 4.3 Column-Based Formats

### 4.3.1 Record Columnar (RC) File Format

This is the most primitive columnar record format that was created as part of the Apache Hive project. RCFile [163] is a binary format similar to SequenceFile that assures high compression for operations that involve multiple rows. Columns are stored as a record in a columnar fashion by creating row splits. Vertical partitions are created on row

splits in a columnar manner. The metadata is stored for each row split as the key and the corresponding data is kept as its value.

#### 4.3.2 Optimized Row Columnar (ORC) File Format

Optimized Row Columnar [153] is similar to Parquet and RCFile in the sense that all these three data file formats exist within the Hadoop framework. The read performance is better. However, writes are slower than average. Besides this, the encoding and compression capabilities of this file format are better than that of its counterparts, with ORC supporting Zlib [164] and Snappy [161].

#### 4.3.3 Parquet

Apache Parquet [154] is a data serialization standard, which is column-oriented and known to improve the efficiency significantly. This data format also includes optimizations like compression on series of values that belong to the same column resulting in improved compaction ratios. Besides this, encodings like bit packing, dictionary and run length encoding are additional optimizations available. In order to compress data, Snappy [161] and GZip [165] algorithms are supported.

#### 4.3.4 CarbonData

This data format was developed by Huawei to manage existing shortcomings in already available formats. CarbonData [166] is a relatively new data file format that allows developers to reap the benefits of column-oriented storage while also providing support for handling random access queries. Data is grouped into blocklets, which is stored alongside other information about the data like schema, indices and offsets, in addition to others. Metadata is stored in headers and footers, which offers significant performance optimization during scanning and processing of subsequent queries.

### 4.4 In-Memory Formats

Apache Arrow [157] is a platform for development of applications using in-memory data. Moreover, it works across languages, which makes it a standard for columnar memory format, enabling support for hierarchical as well as flat data. The data is organized to provide high performance analytics on modern hardware. Interprocess communication and streaming works on zero-copy or no deserialization and serialization. Besides this, it provides many computational libraries for advanced complexities.

### 4.5 Data Storage Services

Kudu [155] is a storage system that ensures scalability and is based on the concept of distributed storage. Data is stored inside tables. Moreover, this data format achieves optimized trade-off between performance and speed of ingestion, which is attributed to the columnar data organization and maintenance of index. Data compression algorithms supported by Apache Kudu include LZ4 [167], Zlib [164] and Snappy [161].

### 4.6 Comparison of Big Data File Formats

The actual storage of data on a file system is determined by the chosen data file format, which is a crucial choice to make in view of the fact that it plays a significant role in optimizing system storage. The decision of choosing a file format for an application depends on the use-case or the algorithm being used for data processing. With that said, the chosen file format must fulfill some basic requirements to be deemed appropriate for big data systems.

Firstly, the chosen big data file format must be expressive and well defined. Secondly, it should have diverse handling capabilities as far as supported data structures are concerned. Some of the basic structures that must be supported include structs, maps, numbers, strings, records and arrays, to name a few. Finally, the big data file format must be binary, simple and provide support for compression.

One of the biggest bottleneck issues in HDFS-related applications that make use of technologies like Spark and Hadoop include reduction in data read and write times. Issues like storage constraints, evolving schemas and big data further complicate the system requirements. In order to mitigate these issues across application domains and problem scenarios, several big data file formats have come into being. The use of an appropriate file format can benefit the system in the following ways –

1. Read time is reduced.
2. Write time is reduced.
3. The files can be split, which in other words means that there is no longer the need to read the whole file for

retrieving a smaller sub-section of it.

4. There is support for schema evolution and schema can be changed on request depending upon the changing needs of the system.
5. There is availability of advanced compression codecs to ensure that files can be compressed without losing the advantages offered by the base format.

In view of the above-mentioned advantages, choosing the right data file format can optimize system performance substantially. However, a plethora of options are available in this regard. While some file formats are developed for general use, there are some others that offer optimization benefits to specific applications or improve specific characteristics. This makes a comparison of the file formats essential for facilitating the decision of which data file format is best suited for an application. Table 3 summarizes the advantages, disadvantages and typical use cases for the different data file formats discussed in the previous sections.

Table 3: Types of Big Data File Formats

| Class                        | Data File Format   | Advantages  | Disadvantages  | Matching Uses  |
|------------------------------|--------------------|---|--|--|
| Text-Based Data File Formats | Text               | 1. Light weight   | 1. Read is slow.<br>2. Write is slow.<br>3. Space is wasted because of column headers that are not required.<br>4. Compressed files cannot be split, which leads to huge maps. Moreover, block-level compression is not supported. | Appropriate for starting use of structured data on HDFS. Also, CSV files are used in cases where data needs to be extracted from Hadoop and bulk-loaded into database.   |
|                              | XML [149]          |   |  |  |
|                              | JSON [151]         |   |  |  |
| Row-Based Data File Format   | SequenceFile [159] | 1. This file format is compact in comparison with Text files.<br>2. The file format supports optional compression.<br>3. It supports parallel processing.<br>4. An enormous number of small-sized files can be stored in a 'container,' provided for the same purpose.<br>5. One of the biggest advantages of this data file format is the support for block-level compression that allows file compression while allowing file splitting for multiple tasks. | 1. This file format is not preferred for tools like Hive.<br>2. The append functionality of the file format just as good and comparable to other file formats.<br>3. The file format lacks support for multiple languages.         | If intermediate data, which is generated between jobs, needs to be saved, then SequenceFile data file format is used.  |
|                              | Avro [156]         | 1. The size of serialized data is smallest.<br>2. It offers block-level compression, allowing file splitting at the same time.<br>3. This file format maintains the structure of the object.<br>4. Even if the schema has changed, Avro allows reading of old data.<br>5. Schema definitions are written in JSON. Therefore, development is considerably simplified in programming languages that possess JSON libraries.                                     | 1. Reading and writing processes need schema definition.   | Avro is considered best for cases where schema evolution is a key requirement. If the schema is expected to change over time, then Avro is preferred. In fact, Avro is preferred for all Hadoop-based use cases. |
| Column-Based                 | RCFile [163]       | RCFile offer typical benefits   | 1. There is no support for schema  | If the use case involves   |

|                            |                  |   |  |   |
|----------------------------|------------------|---|--|---|
| Data File Format           |                  | associated with columnar databases, which include –<br>1. It offers good compression.<br>2. The query performance is better than that for row-oriented databases.   | evolution.<br>2. The write process is slower.  | tables that possess many columns and the application requires frequent use of specific columns, then RCFile is the preferred data format.   |
|                            | ORCFile [153]    | 1. The compression capabilities of ORCFile are better than that of RCFile.<br>2. Query processing is also improved when compared to RCFile.   | 1. There is no support for schema evolution.<br>2. ORCFile format is not well supported by Apache Impala.  | ORCFile and Parquet are used in scenarios where query performance is crucial. However, it has been found that Parquet when used with SparkQL show best improvements in query performance. |
|                            | Parquet [154]    | 1. As is the case with ORCFile format, compression and query performance are good. Moreover, in cases where specific columns are being queried, this data file format is particularly effective.<br>2. The read performance is good.<br>3. Schema evolution, in this case, is better than ORCFile format as columns can be added at the end.<br>4. Storage optimizations are remarkable and it offers file-level as well as block-level compression.  | 1. Writes are computationally intensive.<br>2. If the application requires rows of data, then like all columnar databases, Parquet may not be performance-effective in view of the network activity overheads involved.  |   |
|                            | CarbonData [166] | 1. It supports update and delete operations, which is crucial for many workflows.<br>2. CarbonData offers optimizations like bucketing and multi-layer indexing. Therefore, joining two files is more performance-effective and queries are speedier than ever.   | 1. CarbonData does not support ACID.<br>2. The size of compressed files is larger than those obtained with ORC and Parquet.<br>3. CarbonData is a relatively new data file format with many technologies like Athena [38] and Presto [94] not supporting it yet. |   |
| In-Memory Data File Format | Arrow [157]      | 1. This format enables systems to handle big datasets.<br>2. Same memory can be shared by multiple applications by means of a common data access layer.<br>3. This file format is optimized for parallel processing and data locality. Moreover, it has been developed for Single Instruction Multiple Data (SIMD).<br>4. It allows processing of scan as well as random access workloads.<br>5. The overheads associated with streaming messages and RPC are significantly reduced.<br>6. Apache Arrow can be used | 1. As of now, predicate push down implementation is left to the engine. Although, Apache Arrow is expected to reusable, fast vectorized operations, but efforts in this direction are yet to take shape.   | Apache Arrow is best suited for vectorized processing that involve Single Instruction on Multiple Data (SIMD).  |

|                       |            |  |  |  |
|-----------------------|------------|--|--|--|
|                       |            | with GPUs.<br>7. The columnar format provided by Apache Arrow is ‘fully shredded’ and supports nested and flat schemas.  |  |  |
| Data Storage Services | Kudu [155] | <ol style="list-style-type: none"> <li>1. OLAP workloads can be processed quickly with Kudu.</li> <li>2. Kudu can be seamlessly integrated with Spark, Hadoop and its ecosystem.</li> <li>3. It can be tightly integrated with Apache Impala, which is an effective alternative to Parquet with HDFS.</li> <li>4. The system is highly available.</li> <li>5. The data model provided is structured.</li> <li>6. The management and administration of Kudu is simple.</li> <li>7. The consistency model is flexible and strong.</li> <li>8. Random and sequential workloads can be simultaneously executed with high performance.</li> </ol> | <ol style="list-style-type: none"> <li>1. There are no data restore or backup options inbuilt in Kudu.</li> <li>2. There are some security limitations like authorization available only at the system level and no inbuilt support for data encryption.</li> <li>3. Kudu does not support automatic partitioning and repartitioning of data.</li> <li>4. There are other schema-level limitations like lack of support for secondary indexes and multi-row transactions.</li> </ol> | Kudu has been created for applications centered on time series data and for applications like online reporting and machine data analytics. |

It can be inferred from the comparison table that even though text-based formats are simple and lightweight, they present a host of drawbacks that can affect the performance of the system considerably. In order to overcome the limitations of text-based formats, Hadoop has inbuilt data file formats. The first of these formats is a row-based data file format called SequenceFile [159]. Other data file formats that are based on SequenceFile and are included in the Hadoop ecosystem include MapFile, SetFile and BloomMapFile [168]. These file formats are designed for specialized use cases. SequenceFile only supports Java, which makes it language-dependent, and does not support versioning. As a result, Avro [156], which overpowers SequenceFile, has come up as the most popular row-based data file format. Understandably, Avro is the best option among row-based data file formats.

In case of row-oriented file formats, contiguous storage of rows is done in the file. On the other hand, in case of column-oriented formats, rows are split and values for a row-split are stored column-wise. For example, the values for the first column of a row split are stored first and so on. Therefore, columns not required for a query’s processing can be omitted during data access. In other words, row-oriented formats are best suited for cases in which fewer rows are to be read, but many columns for the row are required. On the other hand, if a small number of columns are required, then column-oriented file formats are a better fit.

It is important to note that column-oriented data file formats require a buffer for row splitting as it works with more than one row at a time. Moreover, it is not possible to control the writing process. If the process fails, it is not possible to recover the current file. This is the reason why column-oriented formats are not used for streaming writes. On the other hand, the use of Avro allows read up to the point until which sync had occurred. Flume makes use of row-oriented formats because of this property [191].

Systems are developed in such a manner that seeks to a disk are kept to a bare minimum and thus, latency is reduced. This is an efficient storage mechanism for transactional workloads for which data is written row-wise. For analytical workloads, large number of rows needs to be accessed at the same time. However, a subset of columns may have to be read for processing a query. In such scenarios, row-oriented format is inefficient considering the fact that all the columns of multiple rows will have to be read even if all these columns are not required. The use of column-oriented format is expected to reduce the number of seeks, improving query performance. Although, writes are slower in this case, but for analytical workloads, this is expected to work well as the number of reads are

expected to outnumber writes.

RCSFile [163] is the most primitive column-based data file format and ORCFile is an optimized version of the same. Although, ORCFile [153] is considered apt for applications with ACID transactions and offers fast access with its indexing feature, Parquet is promoted by Cloudera [169] and is known to perform best with Spark [8]. The most advanced and recent file format in this category is CarbonData [166], but owing to its newer status, its compatibility with technologies is questionable. This makes Parquet the most popular column-based data file format. Arrow and Kudu [155] support columnar representation with the difference that Arrow supports ‘in-memory’ storage while Kudu provisions storage that is mutable on disk as against Parquet format, which is immutable on disk and on disk storage.

The tradeoffs for using immutable data file format (Parquet [154]) include higher throughput for write, easier concurrent sharing, access and replication, and no overheads related to operations. However, for making any modifications, dataset needs to be rewritten. Mutable (Kudu [155]) allows higher flexibility in the compromise between speeds of making updates and reads. The latency for short access is lower owing to quorum replication and primary key indexing. Moreover, the semantics are similar to that of databases. With that said, a separate daemon is required for management.

When comparing Parquet’s on-disk storage with transient in-memory storage of Arrow [157], it is important to note that the former allows multiple query access with priority given to I/O reduction, but CPU throughput must be good. However, on-disk storage is deemed appropriate for streaming access only. On the other hand, in-memory storage supports streaming, as well as random access, and is focused towards execution of a single query. In this case, CPU throughput is prioritized even though I/O throughput must also be good. Some projects make use of Arrow and Parquet together. Pandas [170] is one example of such a usage. The data frame from Pandas can be saved onto Parquet, which can then be read onto Arrow. The ability of Pandas to work on columns of Arrow, allows it to easily integrate with Spark.

## 5.0 Future Trends in Big Data Storage Technology

Technologies used for big data storage have matured over time to give way to next generation technologies like blockchain, which showed a remarkable rise in popularity and usage in a broad collection of domains [171]. As is the case with any technology that is used across domains, several challenges and issues have been identified in its applications to real world scenarios. One of the most profound challenges facing this technology is scalability [172]. However, the capability of this technology to take Internet to its next phase by creating a decentralized web [173] makes it worth a discussion.

One of the fundamental components of the decentralized web is decentralized storage [174]. In addition to blockchain, some other technologies like Internet of Things (IoT) [175] and artificial intelligence [176] are also testing grounds of existing storage solutions and their capabilities. The rise of Internet of Things (IoT) and its expected coverage of 20 billion connected devices [177] by the year 2020 are expected to require acquisition, storage, management and processing of huge data reserves.

The demand from storage solutions is increasing in view of the challenges posed by data management for connected devices, data sharing and need for personalization in applications [178]. There is a shift of approach towards data-intensive applications particularly in view of the dependency of companies on heaps of data. However, this data is stored in centralized data centers, which leads to multiple security breaches and issues [179]. This argument supports the use of decentralized data storage solutions.

However, building decentralized applications on top of blockchain technologies offers its set of independent challenges. For instance, data management and storage are not supported by solutions like Ethereum [180]. Therefore, when these solutions are forced to perform beyond their capacity, they consume larger space and more time. The vision behind decentralized storage is to amalgamate the features of blockchain technology with solutions that can cater to the practical demands of big data storage. It is evident from the name itself that distributed storage divides data into smaller data units and distributes it to different nodes where it is stored. This characteristic can be compared to the distributed ledger technology [181] that is commonly associated with blockchain.

Presently, even cloud-databases are too centralized and easy for hackers to target [182]. Moreover, the points of failure are highly obvious, which makes them all the more, easier to attack. Power outage is one such point of failure [183]. On the contrary, a decentralized system is devoid of such issues in view of the fact that the nodes are geographically distant and physically unrelated. Therefore, an outage or attack on a point of failure is expected to affect only the node concerned. The system will overall remain unaffected, as the other nodes of the system will continue to function as usual. Apart from making the system reliable and available, decentralization also contributes to the scalability and performance of the system [184].

While drawing parallels between decentralized storage and blockchain, it is important to realize that storage on blockchain remains a significant issue [174]. As more and more transactions take place on the blockchain, scalability might become complicated. Therefore, storing big data is certainly a questionable domain from the blockchain perspective. In order to mitigate the challenges mentioned above, techniques like swarming [185] and sharding [186] are being used. Partitioning of databases along logical lines is referred to as sharding. The decentralized model ensures storage of shards together. Moreover, a unique partition key is used by a dedicated decentralized application to access the shards. Besides this, swarming is used to enable collective storage of shards. Data is stored and managed by creating a large group of nodes, which is called a swarm. This group of nodes is similar to the network of nodes created for blockchain.

The most profound advantages of swarming are reduced latency and increased speed [185] considering the fact that data is typically retrieved from the fastest and nearest nodes. In addition to this, the nodes are geographically distant, which makes the system scalable and reliable. Another important aspect of decentralized storage from the customer's point of view is that the customer will have the choice to buy node-level storage from different vendors as against a compulsive single vendor. This can be seen as a major step towards facilitating adoption of this technology in real-world scenarios. It can be concluded that decentralized storage is sure to offer a scalable, efficient and secure solution for the future of big data storage.

## 6.0 Conclusion

Design and development of a big data application that can resolve real world problems and prove to be a viable solution is dependent on the base technologies chosen for the creation of a heterogeneous storage and computing environment. The scope of this research paper is limited to comparison and analysis of storage solutions available for big data systems. There are two facets of storage infrastructures in the big data context. The first facet deals with data storage and management by a dedicated storage infrastructure. In view of specific storage challenges posed by big data like scalability, availability, integration and security, traditional systems are deemed incapable to handle the existing data scenario.

NoSQL has proven to be a viable solution to the big data problem. Few of the significant features of NoSQL that prove the feasibility of its usage for big data storage and management include aspects like easily scalable systems, flexible data modeling, high availability and provisioning of required performance considering the fact that most modern systems handle static as well as real-time data. Features of NoSQL like dynamic schema, auto-sharding, automatic replication and integrated caching abilities mitigate the challenges posed by big data to traditional systems.

Understandably, data is the heart of the system and modeling data is the most crucial design activity for optimum system performance and functionality. Different research works have classified available NoSQL solutions on the basis of supported data models. However, the most accepted classification categorizes NoSQL solutions on the basis of four data models namely document-oriented, graph, key-value and wide-column. The research paper analyzes the benefits and shortcomings of each of these data models to provide an analysis of the use cases that are most appropriate for each data model and the ones that are not. The market is flooded with solutions and technologies that provision a combination of customizable storage, acquisition, processing and visualization [188] facilities to the developer.

This decision is based of many factors that can be technical and non-technical in nature. Once the right data model is determined for a big data problem, a solution that supports the data model along with the desired features

as per the requirements of the big data system need to be found. This research paper provides feature analysis of 80 NoSQL solutions to facilitate decision making in this regard. With that said, benchmarking of these solutions and their performance analysis for a complete quantitative comparison can be performed in the future. Moreover, it has been found that many solutions support multiple data models, making the classification categories insufficient.

It is suggested that hybrid data model categories must be used for discrete classification of solutions to make the decision making process simpler for use-cases that include multi-dimensional data, which might require multiple base data models to design. The second facet of big data storage hails from the fact that different technologies in a heterogeneous technological environment need to share data to operate. The most optimized method for such data sharing is the use of common data file formats. This research paper classifies available big data file formats into five categories namely text-based, row-based, column-based, in-memory and data storage services. Text-based formats have lost utility in the era of big data. However, JSON and XML format usage is common considering the fact that they are lightweight and human readable.

Hadoop provides better ways to store and format data on files. Closest to the relational way of storing data is row-based format. SequenceFile and Avro belong to this category, of which former is the most primitive row-based format provided to users. Other specialized row-based formats that use SequenceFile at their base are also available in Hadoop. These include MapFile, SetFile and BloomMapFile. Avro is the most commonly used row-based format and is preferred for applications that may require all or a majority of the columns.

In most big data scenarios, column-based file formats are known to perform better than their row-based counterparts, as most queries require retrieval of a few columns for multiple rows. Three types of column-based formats are available namely mutable, on-disk storage (Kudu), immutable, on-disk storage (Parquet) and in-memory storage (Arrow). It is important to mention that even though RCFile and ORCFile are also available, Parquet is the most popular column-based format for that category. Some systems use a combination of these formats depending on the requirements of the application. Case study-based performance comparison for quantitative analysis can be performed as future work in this domain.

Besides this, technological decisions are also driven by technical expertise. It has been found that developers accustomed of working on a technology are expected to choose it over solutions that might provide better performance. However, owing to project requirements, they might have to switch to mightier solution, but this wastes development time and effort. Efforts must be made to alleviate such issues. The world of big data storage is drifting from centralized storage to its decentralized counterpart and decentralized storage with blockchain has been identified as the future of big data technology. However, such a synergistic use is faced with several challenges like management of space and time constraints, which must be rigorously dealt with in future work to determine practical feasibility and applicability of such a usage.

## Acknowledgements

This work was supported by a grant from “Young Faculty Research Fellowship” under Visvesvaraya PhD Scheme for Electronics and IT, Department of Electronics & Information Technology (DeitY), Ministry of Communications & IT, Government of India.

## References

- [1] Khan, S., Shakil, K. A., & Alam, M. (2016). Educational intelligence: applying cloud-based big data analytics to the Indian education sector. In *2016 2nd international conference on contemporary computing and informatics (IC3I)* (pp. 29-34). IEEE.
- [2] Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., & Buyya, R. (2015). Big Data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79, 3-15.
- [3] Chen, C. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information sciences*, 275, 314-347.

- [4] George, L. (2011). *HBase: the definitive guide: random access to your planet-size data*. " O'Reilly Media, Inc."
- [5] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The hadoop distributed file system. In *MSSST* (Vol. 10, pp. 1-10).
- [6] Chodorow, K. (2013). *MongoDB: the definitive guide: powerful and scalable data storage*. " O'Reilly Media, Inc."
- [7] Kornacker, M., Behm, A., Bittorf, V., Bobrovitsky, T., Ching, C., Choi, A., ... & Joshi, I. (2015, January). Impala: A Modern, Open-Source SQL Engine for Hadoop. In *Cidr* (Vol. 1, p. 9).
- [8] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. *HotCloud*, 10(10-10), 95.
- [9] Ihaka, R., & Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3), 299-314.
- [10] Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3), 10-20.
- [11] Gu, M., Li, X., & Cao, Y. (2014). Optical storage arrays: a perspective for future big data storage. *Light: Science & Applications*, 3(5), e177.
- [12] White, T. (2012). *Hadoop: The definitive guide*. " O'Reilly Media, Inc."
- [13] Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*, 20.
- [14] Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Ali, M., Kamaleldin, W., ... & Gani, A. (2014). Big data: survey, technologies, opportunities, and challenges. *The Scientific World Journal*, 2014.
- [15] Hausenblas, M., & Nadeau, J. (2013). Apache drill: interactive ad-hoc analysis at scale. *Big Data*, 1(2), 100-104.
- [16] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2), 4.
- [17] Ahmed, S., Ali, M. U., Ferzund, J., Sarwar, M. A., Rehman, A., & Mehmood, A. (2017). Modern data formats for big bioinformatics data analytics. *arXiv preprint arXiv:1707.05364*.
- [18] Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications* (pp. 363-366). IEEE.
- [19] Moniruzzaman, A. B. M., & Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*.
- [20] Namdeo, B., Nagar, N., & Shrivastava, V. (2018). Survey on RDBMS and NoSQL Databases. *International Journal of Innovative Knowledge Concepts*, 6(6), 261-264.
- [21] Sharma, T., Shokeen, V., & Mathur, S. (2018, April). Comparison of Approaches of Distributed Satellite Image Edge Detection on Hadoop. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)* (pp. 645-649). IEEE.
- [22] Rodrigues, M., Santos, M. Y., & Bernardino, J. (2018, October). Experimental Evaluation of Big Data Analytical Tools. In *European, Mediterranean, and Middle Eastern Conference on Information Systems* (pp. 121-127). Springer, Cham.
- [23] Köhler, H., & Link, S. (2018). SQL schema design: foundations, normal forms, and normalization. *Information Systems*, 76, 88-113.
- [24] Rathore, P., Rao, A. S., Rajasegarar, S., Vanz, E., Gubbi, J., & Palaniswami, M. (2018). Real-time urban microclimate analysis using internet of things. *IEEE Internet of Things Journal*, 5(2), 500-511.
- [25] Albayrak, N., Özdemir, A., & Zeydan, E. (2019, February). An Artificial Intelligence Enabled Data Analytics Platform for Digital Advertisement. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)* (pp. 239-241). IEEE.
- [26] Rao, T. R., Mitra, P., Bhatt, R., & Goswami, A. (2018). The big data system, components, tools, and technologies: a survey. *Knowledge and Information Systems*, 1-81.
- [27] Stergiou, C., Psannnis, K. E., Kim, B. G., & Gupta, B. (2018). Secure integration of IoT and cloud

- computing. *Future Generation Computer Systems*, 78, 964-975.
- [28] Lous, P., Tell, P., Michelsen, C. B., Dittrich, Y., & Ebdrup, A. (2018, May). From Scrum to Agile: a journey to tackle the challenges of distributed development in an Agile team. In *Proceedings of the 2018 International Conference on Software and System Process* (pp. 11-20). ACM.
- [29] Cordeiro, J. R., & Postolache, O. (2018, September). Big Data Storage for a Health Predictive System. In *2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*(pp. 1-6). IEEE.
- [30] Zhao, J., Lai, M., Tian, H., & Chang, Y. (2019). *U.S. Patent Application No. 10/205,673*.
- [31] Reddy, K. S., Moharir, S., & Karamchandani, N. (2018, May). Effects of storage heterogeneity in distributed cache systems. In *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)* (pp. 1-8). IEEE.
- [32] Acharya, B., Jena, A. K., Chatterjee, J. M., Kumar, R., & Le, D. N. (2019). NoSQL Database Classification: New Era of Databases for Big Data. *International Journal of Knowledge-Based Organizations (IJKBO)*, 9(1), 50-65.
- [33] Batra, R. (2018). A History of SQL and Relational Databases. In *SQL Primer* (pp. 183-187). Apress, Berkeley, CA.
- [34] Varghese, B., & Buyya, R. (2018). Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79, 849-861.
- [35] Ragmani, A., El Omri, A., Abghour, N., Moussaid, K., & Rida, M. (2018). A performed load balancing algorithm for public Cloud computing using ant colony optimization. *Recent Patents on Computer Science*, 11(3), 179-195.
- [36] Zheng, X. (2018). Database as a Service-Current Issues and Its Future. *arXiv preprint arXiv:1804.00465*.
- [37] Strauch, C. (n.d.). NoSQL Databases. Lecture, Stuttgart Media University.
- [38] Love, P., & Hartland, T. G. (2018). *Using AWS Athena analytics to monitor pilot job health on WLCG compute sites*(No. ATL-SOFT-PROC-2018-059). ATL-COM-SOFT-2018-136.
- [39] Cattell, R. (2011). Scalable SQL and NoSQL data stores. *Acm Sigmod Record*, 39(4), 12-27.
- [40] Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer*, 43(2), 12-14.
- [41] Scofield, B. (2010). NoSQL-Death to Relational Databases. CodeMash Presentation
- [42] Soransso, R. A. S. N., & Cavalcanti, M. C. (2018, April). Data modeling for analytical queries on document-oriented DBMS. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (pp. 541-548). ACM.
- [43] Sun, J., & Milani, C. V. B. (2018). *U.S. Patent Application No. 15/406,643*.
- [44] Geissinger, S. (2018). *U.S. Patent Application No. 15/354,921*.
- [45] Kumar, M. S. (2018). Comparison of NoSQL Database and Traditional Database-An emphatic analysis. *JOIV: International Journal on Informatics Visualization*, 2(2), 51-55.
- [46] Colaso, A., Prieto, P., Herrero, J. A., Abad, P., Menezo, L. G., Puente, V., & Gregorio, J. A. (2018). Memory Hierarchy Characterization of NoSQL Applications through Full-System Simulation. *IEEE Transactions on Parallel and Distributed Systems*, 29(5), 1161-1173.
- [47] Bathla, G., Rani, R., & Aggarwal, H. (2018). Comparative study of NoSQL databases for big data storage. *International Journal of Engineering & Technology*, 7(26), 83.
- [48] da Silva, W. M., Wercelens, P., Walter, M. E. M., Holanda, M., & Brígido, M. (2018, October). Graph Databases in Molecular Biology. In *Brazilian Symposium on Bioinformatics* (pp. 50-57). Springer, Cham.
- [49] Khin, N. T. W., & Yee, N. N. (2018, October). Query Classification based Information Retrieval System. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)* (Vol. 3, pp. 151-156). IEEE.
- [50] Elzein, N. M., Majid, M. A., Hashem, I. A. T., Yaqoob, I., Alaba, F. A., & Imran, M. (2018). Managing big RDF data in clouds: Challenges, opportunities, and solutions. *Sustainable Cities and Society*, 39, 375-386.
- [51] Angles, R., & Gutierrez, C. (2018). An introduction to Graph Data Management. In *Graph Data Management* (pp. 1-32). Springer, Cham.
- [52] Liu, Q., Cao, K., & Bang, F. U. (2018). *U.S. Patent Application No. 10/110,640*.

- [53] Atzeni, P., Cabibbo, L., & Torlone, R. (2018). Data Modeling Across the Evolution of Database Technology. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years* (pp. 221-234). Springer, Cham.
- [54] Chen, J. K., & Lee, W. Z. (2018, December). A study of NoSQL Database for enterprises. In *2018 International Symposium on Computer, Consumer and Control (IS3C)* (pp. 436-440). IEEE.
- [55] Celesti, A., Fazio, M., Romano, A., Bramanti, A., Bramanti, P., & Villari, M. (2018). An oasis-based hospital information system on the cloud: Analysis of a nosql column-oriented approach. *IEEE journal of biomedical and health informatics*, 22(3), 912-918.
- [56] Kamath, S. J., Kanagaratnam, K., Keenleyside, J. D., & Meraji, S. S. (2018). *U.S. Patent No. 9,971,808*. Washington, DC: U.S. Patent and Trademark Office.
- [57] Rudnicki, R., Cox, A. P., Donohue, B., & Jensen, M. (2018, May). Towards a methodology for lossless data exchange between NoSQL data structures. In *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX* (Vol. 10635, p. 106350R). International Society for Optics and Photonics.
- [58] Raj, P., & Deka, G. C. (2018). *A Deep Dive into NoSQL Databases: The Use Cases and Applications* (Vol. 109). Academic Press.
- [59] Ding, X., Chen, L., Gao, Y., Jensen, C. S., & Bao, H. (2018). UItraMan: A unified platform for big trajectory data management and analytics. *Proceedings of the VLDB Endowment*, 11(7), 787-799.
- [60] Miller, A. H., Fisch, A. J., Dodge, J. D., Karimi, A. H., Bordes, A., & Weston, J. E. (2018). *U.S. Patent Application No. 16/002,463*.
- [61] Nguyen, P. A. P., Kryze, D., Vassilakis, T., & Leros, A. (2019). *U.S. Patent Application No. 10/176,236*.
- [62] Das, N., Paul, S., Sarkar, B. B., & Chakrabarti, S. (2019). NoSQL Overview and Performance Testing of HBase Over Multiple Nodes with MySQL. In *Emerging Technologies in Data Mining and Information Security* (pp. 269-279). Springer, Singapore.
- [63] Muñoz-Escobá, F. D., de Juan-Marín, R., García-Escrivá, J. R., González de Mendivil, J. R., & Bernabéu-Aubán, J. M. (2019). CAP Theorem: Revision of Its Related Consistency Models. *The Computer Journal*.
- [64] Rijo, A. D. R. M. (2018). *Building Tunable CRDTs* (Doctoral dissertation).
- [65] Fernandes, D., & Bernardino, J. (2018). Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications, {DATA}* (pp. 373-380).
- [66] Demirci, G. V., & Aykanat, C. (2019). Scaling sparse matrix-matrix multiplication in the accumulo database. *Distributed and Parallel Databases*, 1-32.
- [67] Lee, M., Jeon, S., & Song, M. (2018). Understanding User's Interests in NoSQL Databases in Stack Overflow. In *Proceedings of the 7th International Conference on Emerging Databases* (pp. 128-137). Springer, Singapore.
- [68] Iancu, B., & Georgescu, T. M. (2018). Saving Large Semantic Data in Cloud: A Survey of the Main DBaaS Solutions. *Informatica Economica*, 22(1).
- [69] Cambridge Semantics Inc. (2007). Anzograph. Available from: <https://dbdb.io/db/anzograph>
- [70] Fernandes, D., & Bernardino, J. (2018). Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications, {DATA}* (pp. 373-380).
- [71] Reagan, R. (2018). Azure Data Storage Overview. In *Web Applications on Azure* (pp. 61-76). Apress, Berkeley, CA.
- [72] Gujral, H., Sharma, A., & Kaur, P. (2018, August). Empirical Investigation of Trends in NoSQL-Based Big-Data Solutions in the Last Decade. In *2018 Eleventh International Conference on Contemporary Computing (IC3)* (pp. 1-3). IEEE.
- [73] Ul-Haque, A., Mahmood, T., & Ikram, N. (2018, September). Performance Comparison of State of Art NoSql Technologies Using Apache Spark. In *Proceedings of SAI Intelligent Systems Conference* (pp. 563-576). Springer, Cham.

- [74] Anvari, M., Takht, M. D., & Sefid-Dashti, B. (2018, September). Thrift Service Composition: Toward Extending BPEL. In *Proceedings of the international conference on smart cities and internet of things* (p. 13). ACM.
- [75] Maity, B., Acharya, A., Goto, T., & Sen, S. (2018, June). A Framework to Convert NoSQL to Relational Model. In *Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology* (pp. 1-6). ACM.
- [76] Daniel, G., Sunyé, G., & Cabot, J. (2018). Advanced prefetching and caching of models with PrefetchML. *Software & Systems Modeling*, 1-22.
- [77] Gupta, S., & Narsimha, G. (2018). Miscegenation of scalable and DEP3K performance evaluation of nosql-cassandra for bigdata applications deployed in cloud. *International Journal of Business Process Integration and Management*, 9(1), 12-21.
- [78] Hung, H., Rajamani, K., Lee, J., Jain, S., Ravipati, G., McHugh, J., ... & Huang, J. C. (2019). U.S. Patent Application No. 16/135,769.
- [79] Iancu, B., & Georgescu, T. M. (2018). Saving Large Semantic Data in Cloud: A Survey of the Main DBaaS Solutions. *Informatica Economica*, 22(1).
- [80] Mahmood, A. A. (2018). Automated Algorithm for Data Migration from Relational to NoSQL Databases. *ALNAHRAIN JOURNAL FOR ENGINEERING SCIENCES*, 21(1), 60-65.
- [81] Lathar, P., Srinivasa, K. G., Kumar, A., & Siddiqui, N. (2018). Comparison Study of Different NoSQL and Cloud Paradigm for Better Data Storage Technology. In *Handbook of Research on Cloud and Fog Computing Infrastructures for Data Science* (pp. 312-343). IGI Global.
- [82] Chang, J., Gutsche, O., Mandrichenko, I., & Pivarski, J. (2018, September). Striped Data Server for Scalable Parallel Data Analysis. In *Journal of Physics: Conference Series* (Vol. 1085, No. 4, p. 042035). IOP Publishing.
- [83] Smith, Z. (2018). Joining and aggregating datasets using CouchDB (Doctoral dissertation, University of Cape Town).
- [84] Kepner, J., Gadepally, V., Milechin, L., Samsi, S., Arcand, W., Bestor, D., ... & Jones, M. (2019). A Billion Updates per Second Using 30,000 Hierarchical In-Memory D4M Databases. arXiv preprint arXiv:1902.00846.
- [85] Reagan, R. (2018). Cosmos DB. In *Web Applications on Azure* (pp. 187-255). Apress, Berkeley, CA.
- [86] Angles, R., Arenas, M., Barceló, P., Boncz, P., Fletcher, G., Gutierrez, C., ... & van Rest, O. (2018, May). G-CORE: A core for future graph query languages. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 1421-1432). ACM.
- [87] Noghabi, S. A., Kolb, J., Bodik, P., & Cuervo, E. (2018). Steel: Simplified development and deployment of edge-cloud applications. In *10th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18)*.
- [88] Montella, R., Di Luccio, D., Kosta, S., Giunta, G., & Foster, I. (2018, October). Performance, resilience, and security in moving data from the fog to the cloud: The DYNAMO transfer framework approach. In *International Conference on Internet and Distributed Computing Systems* (pp. 197-208). Springer, Cham.
- [89] Seda, P., Hosek, J., Masek, P., & Pokorny, J. (2018, April). Performance testing of NoSQL and RDBMS for storing big data in e-applications. In *2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG)* (pp. 1-4). IEEE.
- [90] Iancu, B., & Georgescu, T. M. (2018). Saving Large Semantic Data in Cloud: A Survey of the Main DBaaS Solutions. *Informatica Economica*, 22(1).
- [91] Sánchez-de-Madariaga, R., Muñoz, A., Castro, A. L., Moreno, O., & Pascual, M. (2018). Executing Complexity-Increasing Queries in Relational (MySQL) and NoSQL (MongoDB and EXist) Size-Growing ISO/EN 13606 Standardized EHR Databases. *JoVE (Journal of Visualized Experiments)*, (133), e57439.
- [92] Chrysafis, C., Collins, B., Dugas, S., Dunkelberger, J., Ehsan, M., Gray, S., ... & McMahan, M. (2019). FoundationDB Record Layer: A Multi-Tenant Structured Datastore. arXiv preprint arXiv:1901.04452.
- [93] Imran, M., Ahamad, M. V., Haque, M., & Shoaib, M. (2018). Big Data Analytics Tools and Platform in Big Data Landscape. In *Handbook of Research on Pattern Engineering System Development for Big Data*

- Analytics (pp. 80-89). IGI Global.
- [94] Pant, P., Kumar, P., Alam, I., & Rawat, S. (2018). Analytical Planning and Implementation of Big Data Technology Working at Enterprise Level. In *Information Systems Design and Intelligent Applications* (pp. 1031-1043). Springer, Singapore.
- [95] Yuzuk, S., Aktas, M. G., & Aktas, M. S. (2018, December). On the Performance Analysis of Map-Reduce Programming Model on In-Memory NoSQL Storage Platforms: A Case Study. In 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) (pp. 45-50). IEEE.
- [96] Jayagopal, V., & Basser, K. K. (2019). Data Management and Big Data Analytics: Data Management in Digital Economy. In *Optimizing Big Data Management and Industrial Systems With Intelligent Techniques* (pp. 1-23). IGI Global.
- [97] Vonitsanos, G., Kanavos, A., Mylonas, P., & Sioutas, S. (2018, July). A NoSQL Database Approach for Modeling Heterogeneous and Semi-Structured Information. In 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA) (pp. 1-8). IEEE.
- [98] Marinov, M., Georgiev, G., & Popova, E. (2018, May). NoSQL approach for sensor data storage and retrieval. In 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1427-1432). IEEE.
- [99] Kondratenko, Y., Kondratenko, G., & Sidenko, I. (2018, May). Multi-criteria decision making for selecting a rational IoT platform. In 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT) (pp. 147-152). IEEE.
- [100] IBM. (n.d.) IBM Informix C-ISAM. Available from: <https://www.ibm.com/in-en/marketplace/ibm-informix-cisam>
- [101] Acharya, S. (2018). *Apache Ignite Quick Start Guide: Distributed data caching and processing made easy*. Packt Publishing Ltd.
- [102] González-Aparicio, M. T., Younas, M., Tuya, J., & Casado, R. (2018). Testing of transactional services in NoSQL key-value databases. *Future Generation Computer Systems*, 80, 384-399.
- [103] Płaza, M., Deniziak, S., Płaza, M., Belka, R., & Pięta, P. (2018, October). Analysis of parallel computational models for clustering. In *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018* (Vol. 10808, p. 108081O). International Society for Optics and Photonics.
- [104] Agarwal, M., & Srivastava, G. M. S. (2019). “Big” Data Management in Cloud Computing Environment. In *Harmony Search and Nature Inspired Optimization Algorithms* (pp. 707-716). Springer, Singapore.
- [105] Kovács, T., Simon, G., & Mezei, G. (2018, June). Benchmarking Graph Database Backends—What Works Well with Wikidata?. In *THE 11TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE* (p. 154).
- [106] Ahmad, K., Alam, M. S., & Udzir, N. I. (2019). Security of NoSQL Database Against Intruders. *Recent Patents on Engineering*, 13(1), 5-12.
- [107] Erraissi, A., & Belangour, A. (2018, December). Meta-modeling of Zookeeper and MapReduce processing. In *2018 International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)* (pp. 1-5). IEEE.
- [108] Qader, M. A., Cheng, S., & Hristidis, V. (2018, May). A comparative study of secondary indexing techniques in LSM-based NoSQL databases. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 551-566). ACM.
- [109] Johnsirani Venkatesan, N., Nam, C., & Shin, D. R. (2019). Deep Learning Frameworks on Apache Spark: A Review. *IETE Technical Review*, 36(2), 164-177.
- [110] DelGaudio, C. I., Hicks, S. D., Houston, W. M., Kurtz, R. S., Hanrahan, V. A., Martin Jr, J. A., ... & Rauch, D. C. (2018). U.S. Patent No. 9,928,480. Washington, DC: U.S. Patent and Trademark Office.
- [111] Eshtay, M., Sleit, A., & Aldwairi, M. (2019). IMPLEMENTING BI-TEMPORAL PROPERTIES INTO VARIOUS NOSQL DATABASE CATEGORIES. *International Journal of Computing*, 18(1), 45-52.
- [112] Li, J., & Li, J. (2018, June). Research on NoSQL Database Technology. In 2018 2nd International

- Conference on Management, Education and Social Science (ICMESS 2018). Atlantis Press.
- [113] Patel, Y., Verma, M., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2018). Revisiting concurrency in high-performance NoSQL databases. In 10th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 18).
- [114] Flores, A., Ramírez, S., Toasa, R., Vargas, J., Urvina-Barrionuevo, R., & Lavin, J. M. (2018, April). Performance Evaluation of NoSQL and SQL Queries in Response Time for the E-government. In 2018 International Conference on eDemocracy & eGovernment (ICEDEG) (pp. 257-262). IEEE.
- [115] Chutea, C. G., & Huffb, S. M. (2018, January). The pluripotent rendering of clinical data for precision medicine. In MEDINFO 2017: Precision Healthcare Through Informatics: Proceedings of the 16th World Congress on Medical and Health Informatics (Vol. 245, p. 337). IOS Press.
- [116] Webber, J., & Robinson, I. (2018). A programmatic introduction to neo4j. Addison-Wesley Professional.
- [117] BigOpenData.eu. (2017). NoSQLz. Available from: <https://www.bigopendata.eu/tag/nosqlz-developers/>
- [118] EkkySoftware. (2012). ObjectDatabase++. Available from: <https://web.archive.org/web/20120926131201/http://www.ekkysoftware.com/ODBPP>
- [119] Yao, J. (2018). A comparison of different graph database types.
- [120] Bugiotti, F. (2018, September). Modeling Strategies for Storing Data in Distributed Heterogeneous NoSQL Databases. In Conceptual Modeling: 37th International Conference, ER 2018, Xi'an, China, October 22–25, 2018, Proceedings (Vol. 11157, p. 488). Springer.
- [121] Agrawal, D., Ganti, R. K., Lee, K., & Srivatsa, M. (2018). U.S. Patent No. 9,886,785. Washington, DC: U.S. Patent and Trademark Office.
- [122] Yamaguchi, T., Brain, M., Ryder, C., Imai, Y., & Kawamura, Y. (2019, January). Application of Abstract Interpretation to the Automotive Electronic Control System. In International Conference on Verification, Model Checking, and Abstract Interpretation (pp. 425-445). Springer, Cham.
- [123] Makris, A., Tserpes, K., Spiliopoulos, G., & Anagnostopoulos, D. (2019). Performance Evaluation of MongoDB and PostgreSQL for spatio-temporal data.
- [124] Banane, M., & Belangour, A. (2018, July). A Survey on RDF Data Store Based on NoSQL Systems for the Semantic Web Applications. In International Conference on Advanced Intelligent Systems for Sustainable Development (pp. 444-451). Springer, Cham.
- [125] Plaza, M., Deniziak, S., Plaza, M., Belka, R., & Pięta, P. (2018, October). Analysis of parallel computational models for clustering. In Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018 (Vol. 10808, p. 108081O). International Society for Optics and Photonics.
- [126] Eini, O. (2018). Inside RavenDB 4.0.
- [127] Diogo, M., Cabral, B., & Bernardino, J. (2019). Consistency Models of NoSQL Databases. *Future Internet*, 11(2), 43.
- [128] Wingerath, W., Gessert, F., Witt, E., Friedrich, S., & Ritter, N. (2018). Real-Time Data Management for Big Data. In EDBT (pp. 524-527).
- [129] Kumar, M. S. (2018). Comparison of NoSQL Database and Traditional Database-An emphatic analysis. *JOIV: International Journal on Informatics Visualization*, 2(2), 51-55.
- [130] Rocket (n.d.). RocketU2. Available from: <https://www.rocketsoftware.com/products/rocket-u2/documentation>
- [131] Papaioannou, A., & Magoutis, K. (2018, July). Replica-group leadership change as a performance enhancing mechanism in NoSQL data stores. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS) (pp. 1448-1453). IEEE.
- [132] Hwang, M. (2018). Graph Processing Using SAP HANA: A Teaching Case. *e-Journal of Business Education and Scholarship of Teaching*, 12(2), 155-165.
- [133] Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. *ACM Computing Surveys (CSUR)*, 51(2), 40.
- [134] Segeljakt, K. (2018). A Scala DSL for Rust code generation.

- [135] Swami, D., & Sahoo, B. (2018). Storage size estimation for schemaless big data applications: A JSON-based overview. In *Intelligent Communication and Computational Technologies* (pp. 315-323). Springer, Singapore.
- [136] Martins, P., Abbasi, M., & Sá, F. (2019, April). A Study over NoSQL Performance. In *World Conference on Information Systems and Technologies* (pp. 603-611). Springer, Cham.
- [137] Deka, G. C. (2018). NoSQL Web Crawler Application. In *Advances in Computers* (Vol. 109, pp. 77-100). Elsevier.
- [138] Franciscus, N., Ren, X., & Stantic, B. (2018). Precomputing architecture for flexible and efficient big data analytics. *Vietnam Journal of Computer Science*, 5(2), 133-142.
- [139] Hu, K., & Zhu, J. (2018, October). A Progressive Web Application on Ancient Roman Empire Coins and Relevant Historical Figures with Graph Database. In *Euro-Mediterranean Conference* (pp. 235-241). Springer, Cham.
- [140] Kalyonova, O., Akparaliev, N., & Perl, I. (2018, November). Design Of Specialized Storage for Heterogeneous Project Data. In *Proceedings of the 23rd Conference of Open Innovations Association FRUCT* (p. 21). FRUCT Oy.
- [141] Malki, M. E. L., Hamadou, H. B., El Malki, N., & Kopliku, A. (2018). MPT: suite tools to support performance tuning in NoSQL systems.
- [142] Yin, X., & Luo, Q. (2018, April). Research and Application of Large Data Query Technology Based on NoSQL Database. In *2018 3rd International Workshop on Materials Engineering and Computer Sciences (IWMECS 2018)*. Atlantis Press.
- [143] Rai, R., & Chettri, P. (2018). NoSQL Hands On. In *Advances in Computers* (Vol. 109, pp. 157-277). Elsevier.
- [144] Fernandes, K., Melhem, R., & Hammoud, M. (2018, December). Dynamic Elasticity for Distributed Graph Analytics. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (pp. 145-148). IEEE.
- [145] Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., ... & Taylor, A. (2018, May). Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 1433-1445). ACM.
- [146] Rahman, H. U., Khan, R. U., & Ali, A. (2018). Programming and Pre-Processing Systems for Big Data Storage and Visualization. In *Handbook of Research on Big Data Storage and Visualization Techniques* (pp. 228-253). IGI Global.
- [147] Mishra, R. K. (2018). The Era of Big Data, Hadoop, and Other Big Data Processing Frameworks. In *PySpark Recipes* (pp. 1-14). Apress, Berkeley, CA.
- [148] Bansal, K., Chawla, P., & Kurle, P. (2019). Analyzing Performance of Apache Pig and Apache Hive with Hadoop. In *Engineering Vibration, Communication and Information Processing* (pp. 41-51). Springer, Singapore.
- [149] Asemota, E., Gallagher, S., Mcrobbie, G., & Cochran, S. (2018). Defining case based reasoning cases with xml.
- [150] Gallego, J. J. C., & Fernández-Bermejo, M. D. M. (2018). Analysis of the impact of file formats for open data analytics efficiency: a case study with R. *GSTF Journal on Computing (JoC)*, 5(1).
- [151] Rebelo Moreira, J. L., Ferreira Pires, L., & Van Sinderen, M. (2018). Semantic Interoperability for the IoT: Analysis of JSON for Linked Data. *Enterprise Interoperability: Smart Services and Business Impact of Enterprise Interoperability*, 163-169.
- [152] Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. *Wireless Communications and Mobile Computing, 2018*.
- [153] Gupta, A., Saxena, M., & Gill, R. (2018, April). Performance Analysis of RDBMS and Hadoop Components with Their File Formats for the Development of Recommender Systems. In *2018 3rd International Conference for Convergence in Technology (I2CT)* (pp. 1-6). IEEE.

- [154] Gershinsky, G. (2018, June). Efficient Analytics on Encrypted Data. In *Proceedings of the 11th ACM International Systems and Storage Conference* (pp. 121-121). ACM.
- [155] Quinto, B. (2018). *Next-Generation Big Data: A Practical Guide to Apache Kudu, Impala, and Spark*. Apress.
- [156] Hukill, G. S., & Hudson, C. (2018). Avro: Overview and Implications for Metadata Processing.
- [157] White, R. M. (2018). Open Data Standards for Administrative Data Processing.
- [158] Carter, P. A. (2018). Understanding JSON. In *SQL Server Advanced Data Types* (pp. 181-200). Apress, Berkeley, CA.
- [159] Ahad, M. A., & Biswas, R. (2019). Handling Small Size Files in Hadoop: Challenges, Opportunities, and Review. In *Soft Computing in Data Analytics* (pp. 653-663). Springer, Singapore.
- [160] Xue, Y., Tan, Y. A., Liang, C., Zhang, C., & Zheng, J. (2018). An optimized data hiding scheme for deflate codes. *Soft Computing*, 22(13), 4445-4455.
- [161] Sansanwal, K., Shrivastava, G., Anand, R., & Sharma, K. (2019). Big Data Analysis and Compression for Indoor Air Quality. *Handbook of IoT and Big Data*, 1.
- [162] Carroll, J. R., & Robertson, F. R. (2019). *A COMPARISON OF PHASOR COMMUNICATIONS PROTOCOLS* (No. PNNL-28499). Pacific Northwest National Lab.(PNNL), Richland, WA (United States).
- [163] Makki, S. K., & Hasan, M. R. (2018). Measuring the Performance of Data Placement Structures for MapReduce-based Data Warehousing Systems. *International Journal of New Computer Architectures and Their Applications*, 8(1), 11-21.
- [164] Rhu, M., O'Connor, M., Chatterjee, N., Pool, J., Kwon, Y., & Keckler, S. W. (2018, February). Compressing DMA engine: Leveraging activation sparsity for training deep neural networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 78-91). IEEE.
- [165] Kösters, M., Leufken, J., Schulze, S., Sugimoto, K., Klein, J., Zahedi, R. P., ... & Fufezan, C. (2018). pymzML v2. 0: introducing a highly compressed and seekable gzip format. *Bioinformatics*, 34(14), 2513-2514.
- [166] Jiang, H., & Elmore, A. J. (2018, June). Boosting data filtering on columnar encoding with SIMD. In *Proceedings of the 14th International Workshop on Data Management on New Hardware* (p. 6). ACM.
- [167] Liu, W., Mei, F., Wang, C., O'Neill, M., & Swartzlander, E. E. (2018). Data compression device based on modified LZ4 algorithm. *IEEE Transactions on Consumer Electronics*, 64(1), 110-117.
- [168] Tchaye-Kondi, J., Zhai, Y., Lin, K. J., Tao, W., & Yang, K. (2019). Hadoop Perfect File: A fast access container for small files with direct in disc metadata access. *arXiv preprint arXiv:1903.05838*.
- [169] Ahuja, R. (2018). Hadoop Framework for Handling Big Data Needs. In *Handbook of Research on Big Data Storage and Visualization Techniques* (pp. 101-122). IGI Global.
- [170] Peltenburg, J., van Straten, J., Brobbel, M., Hofstee, H. P., & Al-Ars, Z. (2019, April). Supporting Columnar In-memory Formats on FPGA: The Hardware Design of Fletcher for Apache Arrow. In *International Symposium on Applied Reconfigurable Computing* (pp. 32-47). Springer, Cham.
- [171] Kaur, H., Alam, M. A., Jameel, R., Mourya, A. K., & Chang, V. (2018). A proposed solution and future direction for blockchain-based heterogeneous medicare data in cloud environment. *Journal of medical systems*, 42(8), 156.
- [172] Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: a survey. *International Journal of Web and Grid Services*, 14(4), 352-375.
- [173] Lai, Z., Liu, C., Lo, E., Kao, B., & Yiu, S. M. (2018). Decentralized Search on Decentralized Web. *arXiv preprint arXiv:1809.00939*.
- [174] Wang, S., Zhang, Y., & Zhang, Y. (2018). A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *IEEE Access*, 6, 38437-38450.
- [175] Ray, P. P. (2018). A survey on Internet of Things architectures. *Journal of King Saud University-Computer and Information Sciences*, 30(3), 291-319.
- [176] Lu, H., Li, Y., Chen, M., Kim, H., & Serikawa, S. (2018). Brain intelligence: go beyond artificial

- intelligence. *Mobile Networks and Applications*, 23(2), 368-375.
- [177] Tripathi, R. (2018). Enabling 20 Billion Devices Through The Internet Of Things. Available from: <https://www.digitalistmag.com/iot/2018/03/02/enabling-20-billion-devices-through-internet-of-things-05944011>
- [178] Khan, S., Shakil, K. A., & Alam, M. (2018). Cloud-based big data analytics—a survey of current research and future directions. In *Big Data Analytics* (pp. 595-604). Springer, Singapore.
- [179] Paralkar, K., Yadav, S., Kumari, S., Kulkarni, A., & Pingat, S. P. (2018). Photogroup: Decentralized Web Application Using Ethereum Blockchain. *International Research Journal of Engineering and Technology*, 5 (4), 489-492.
- [180] Akira, S. (2018). Ethereum-The Next Generation of Cryptocurrency: A Guide to the World of Ethereum.
- [181] Siano, P., De Marco, G., Rolán, A., & Loia, V. (2019). A Survey and Evaluation of the Potentials of Distributed Ledger Technology for Peer-to-Peer Transactive Energy Exchanges in Local Energy Markets. *IEEE Systems Journal*.
- [182] Ansar, M., & Fatima, M. (2018). Biometric Encryption in Cloud Computing: A Systematic Review. *International Journal of Computer Science and Network Solutions*, 6(1), 10-19.
- [183] Shi, D., Chen, X., Wang, Z., Zhang, X., Yu, Z., Wang, X., & Bian, D. (2018). A distributed cooperative control framework for synchronized reconnection of a multi-bus microgrid. *IEEE Transactions on Smart Grid*, 9(6), 6646-6655.
- [184] Novo, O. (2018). Blockchain meets IoT: An architecture for scalable access management in IoT. *IEEE Internet of Things Journal*, 5(2), 1184-1195.
- [185] Ferrer, E. C. (2018, November). The blockchain: a new framework for robotic swarm systems. In *Proceedings of the Future Technologies Conference* (pp. 1037-1058). Springer, Cham.
- [186] Dang, H., Dinh, T. T. A., Loghin, D., Chang, E. C., Lin, Q., & Ooi, B. C. (2018). Towards Scaling Blockchain Systems via Sharding. *arXiv preprint arXiv:1804.00399*.
- [187] Khan, S., Liu, X., Shakil, K. A., & Alam, M. (2017). A survey on scholarly data: From big data perspective. *Information Processing & Management*, 53(4), 923-944.
- [188] Khan, S., Shakil, K. A., Alam, M. (2017). Big Data Computing Using Cloud-Based Technologies: Challenges and Future Perspectives In: *Networks of the Future: Architectures, Technologies, and Implementations*. Chapman and Hall/CRC.
- [189] Wilcox, T., Jin, N., Flach, P., & Thumim, J. (2019). A Big Data platform for smart meter data analytics. *Computers in Industry*, 105, 250-259.
- [190] Chen, J. K., & Lee, W. Z. (2018, December). A study of NoSQL Database for enterprises. In *2018 International Symposium on Computer, Consumer and Control (IS3C)* (pp. 436-440). IEEE.
- [191] Kovačević, I., & Mekterovic, I. (2018, May). Novel BI data architectures. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1191-1196). IEEE.
- [192] Pal, D., Triyason, T., & Padungweang, P. (2018). Big Data in Smart-Cities: Current Research and Challenges. *Indonesian Journal of Electrical Engineering and Informatics (IJEI)*, 6(4), 351-360.
- [193] Portella, G., Rodrigues, G. N., Nakano, E., & Melo, A. C. (2018). Statistical analysis of Amazon EC2 cloud pricing models. *Concurrency and Computation: Practice and Experience*, e4451.
- [194] Daher, Z., & Hajjdiab, H. (2018). Cloud Storage Comparative Analysis Amazon Simple Storage vs. Microsoft Azure Blob Storage. *International Journal of Machine Learning and Computing*, 8(1).
- [195] Khobragade, S. V., Nalbalwar, S. L., & Nandgaonkar, A. B. (2018). Fusion Execution of NaCl on Tree-Shaped MSA. *International Journal of Antennas and Propagation*, 2018.
- [196] Bielefeldt, A., Gonsior, J., & Krötzsch, M. (2018). Practical linked data access via SPARQL: the case of wikidata. In *Proc. WWW2018 Workshop on Linked Data on the Web (LDOW-18)*. *CEUR Workshop Proceedings, CEUR-WS.org*.
- [197] Thakkar, H., Punjani, D., Keswani, Y., Lehmann, J., & Auer, S. (2018). A Stitch in Time Saves Nine--SPARQL querying of Property Graphs using Gremlin Traversals. *arXiv preprint arXiv:1801.02911*.