flexibility in data types and allows easier implementation of the 3D models. Unstructured databases have schema-less features and a simple interface that enables developers to develop databases quickly. It is worth mentioning that this paper's objective is to analyze the four categories of the unstructured database, identify unstructured databases capable of handling 3D point cloud and provide perspectives in the field of 3D point cloud handling. This paper offers an alternative way for the developers to store 3D point cloud as there need to utilize an efficient and scalable geospatial database which strongly influences the overall performance of the applications. In addition, this paper's contribution is to guide developers in choosing the appropriate database.

The rest of the paper is organized as follows. Section 2 explores related reviews and benchmarking tools used to compare unstructured databases. Section 3 describes some preliminary concepts of 4 types of unstructured databases: document store, key-value, column store, and graph store database, starting with the description and analysis of the different implementations of databases available on the market. Section 4 analysed using a 3D point cloud in the unstructured database. Section 5 analysed an unstructured database based on three main concepts: data model, querying, and scaling. Finally, section 6 discusses the application of unstructured databases and conclusions on the findings.

2. RELATED WORKS

Previous study shows several attempts on managing large size of 3D data using relational database or unstructured database. For example, (Azri et al. 2020) proposed a 3D data structure for fast retrieval nearest neighbour search in 3D urban environment. However, current relational database is proved to be ineffective while dealing with large volume of data such as point cloud. Many works list the comparison between relational and unstructured databases, revealing their virtues and weakness. However, there are very few works describe how to organize 3D point clouds in unstructured databases.

Research by (Grolinger, Higashino et al. 2013) titled Data Management in Cloud Environments: NoSQL and NewSQL Data Stores, focused on the storage aspect of cloud computing systems, particular in NoSQL and NewSQL data stores. The results are alternatives to traditional relational databases that can handle huge volumes of data by utilising the cloud environment. In addition, the research reviewed NoSQL and NewSQL data stores to provide a perspective and guidance to practitioners and researchers in choosing appropriate storage solutions and identifying challenges and opportunities in the NoSQL field. Thus, from this work, challenges in the domain, including terminology diversity and inconsistency, limited documentation, sparse comparison and benchmarking criteria, occasional immaturity of solutions and lack of support, and non-existence of a standard query language, can be identified study for future research. This research has helped give general knowledge and introduction to unstructured databases and their challenges.

Furthermore, research conducted (Višnjevac, Mihajlović et al. 2019) on an approach of using an unstructured database and JavaScript application for 3D visualisation stated that an unstructured database can be used for storing 3D cadastral data defined by a data model based on LADM. BASE principles protect the stored 3D cadastral data and developed JavaScript applications can easily meet basic 3D cadastral requirements for 3D visualisation, such as viewing and selecting each 3D object. However, no topology validation is done on the model, reducing the data integrity. Furthermore, more intuitive visualisation approaches are required to enable simpler system use for professionals and non-professionals. The prototype is a good starting point for developing a modern cadastral system that enables 3D registration of real properties, facilitates unequivocal registration of complex 3D property situations, and provides an intuitive user interface and 3D spatial data visualisation methods.

Then (Liu and Boehm 2015) presented a file-centric storage and retrieval system for large collections of LiDAR point cloud tiles based on scalable NoSQL technology 2015. The system can store large collections of point clouds and use a document-based unstructured database to retain a file-centric workflow, making many existing tools accessible. The suggested system supports spatial queries on the tile geometry. Compared to file system operation, it has several overheads on inserting and retrieving the files locally, but the remote transfer is at par with popular cloud storage. It uses a MongoDB application that supports Map-Reduce internally for database queries and provides a special adapter to access MongoDB from Hadoop. However, this research only focuses on data storage and retrieval. The mentioned method can be used as a reference for managing 3D spatial objects in an unstructured database.

Lastly, (Reniers, Van Landuyt et al. 2019) research identifies 341 frameworks relevant to object-relational impedance mismatch and evaluate 11 Object-NoSQL data mappers (ONDMs) in detail. The research investigated the ONDMs in terms of their mapping strategies for collections, relationships and inheritance and compared available mapping strategies systematically. The study concludes that the collections and object-owned data are typically embedded by default, and support for alternate strategies is lacking. Relationships can be embedded within the referring object as a set of referred keys or identifiers and embedding of relationship data is scarcely supported. The results show that object-document mapping strategies are more sophisticated and advanced than column and graph stores. The presented survey is an important steppingstone in our ongoing research on NoSQL abstractions, appropriate mapping strategies and schema design, and optimised configuration and deployment of highly distributed databases. Thus, the framework of this study can be used to examine fundamental mapping issues between objects and unstructured databases. This research helps manage objects in the database.

3. 3D POINT CLOUD MANAGEMENT IN AN UNSTRUCTURED DATABASE

The 3D point cloud is a set of data points in space that

represent 3D space or an object. It contains rich geographic information such as three-dimensional coordinates, RGB information, colour and reflection strength (Xu and Guo 2016). MongoDB and HBase are the most popular unstructured database applications for 3D point cloud handling. This can be seen in a study by (Azri, Ujang et al. 2021), MongoDB uses outpaces relational databases during the data retrieval and updating of the 3D point cloud. Furthermore, the study also visualises the point cloud from the database into PotreeViewer. (Višnjevac, Mihajlović et al. 2019) created a prototype of the 3D cadastral system based on the unstructured database and JavaScript application for storing and visualisation. (Zhang, Wang et al. 2019) use HBase to distribute architecture for massive remote sensing data based on the metadata file, pyramid model, and Hilbert curve. It is proved efficient compared to the traditional database. (Xu and Guo 2016) established a sharding cluster for MongoDB under a distributed environment while distributing storage, spatial query, and MapReduce for point cloud data through scope sharding, and Hash-based sharding successfully reflects huge advantages of MongoDB in storage and processing for geospatial data. Then, (Luan, Yachun et al. 2014) proposed a 3D model management strategy based on Hadoop to deal with the big data problem of 3D model management. (Park, Yun et al. 2011) used MapReduce and utilised the unstructured database to increase the analysis velocity. It is suitable for geospatial big data applications such as 3D point cloud overview and visualisation.

Unstructured database systems are now an efficient way to manage massive data volumes distributed across numerous servers (Atzeni, Bugiotti et al. 2020). It is appropriate for big data applications like the generalisation and visualisation of 3D city models (Mao, Harrie et al. 2014). There are commonly 4 types of model stores in unstructured databases (Li 2018). The data store is divided into document-store, key-value store, column-store, and graph-store categories based on their data models (Cattell 2011).

The Document store unstructured database stores the data in a collection of key-value pairs that are compressed as a document and required its values stored, providing structure, and encoding the metadata. All document keys must be unique, and every document contains a unique key of "ID" that unique in the collection of documents and identifies a document unambiguously. Document stores also did not restrict the schema and can support multi-attribute search records containing various key-value pairs (Makris, Tserpes et al. 2016). The document store is more efficient in managing geospatial data than key-value because the geospatial data inside the document store can be retrieved using flexible queries. In multiple cases, they can be used in storing and handling geospatial data, thus allowing some queries such as proximity to be implemented. (Amirian, Basiri et al. 2014). As document store does not support relationships and join the way relational databases are, the document store database nature has a major effect on the data retrieval. For instance, when the database stores millions of 3D point cloud data inside a collection, the queries will be very fast as it is stored in a document containing the same schema.

Key-Values Stores stored every data item as a key-value pair in schema-less format. A key is a string type unique ID that points to the data with which it is associated that stored values in any data type or without any predefined schema of BLOB (Binary Large Object). The values are stored as uninterrupted byte arrays, thus only allowing the key as the only route for data retrieval. The key-value pair is grouped into a collection to structure the data model. Thus, allowing it to support simple query operations that are applied on the attribute key only. The key-value stores all the data in-memory key-value stores (IMKVS), making them great databases for massive 3D point cloud data as the data is stored in the cache and contains high throughput system requirements. The application is responsible for understanding the object type and parsing it accordingly.

Column Stores maintain data as a set of columns and data distribution based on columns instead of rows, an arrangement usually enhanced for queries throughout large datasets. The column has the smallest data unit, containing a key, value, and timestamp. Besides the column name, the database stores all values, and the null values are stored by ignoring the column. Columns that are related to each other create a column family. All the data in a single column family is stored in a physical set of files and sorted in chronological order. This makes it easier to support versioning and achieve performance. Thus, providing higher performance on data retrieval and query operation. Querying over rows is memory intensive and requires huge disk access, particularly when each row comprises many columns. In a column store, columns are grouped into column families, and each column family can have an unlimited number of columns, and in Cassandra a super column is a column containing other columns but cannot include other super columns. The column family aids in supporting organisation and partitioning. This way, it is easier to query the entire collection of columns for all the rows.

Graph-store stores network data consisting of nodes (entities), edges (relationships) and properties. Various nodes might contain multiple properties. Graph stores are suitable for network data models such as social networks, road networks, and IT networks, where entities are highly connected through relationships. The graph store highlights that each node contains a direct indication of its adjoining node. Thus, index lookups are not required for navigating data connected. Consequently, the massive amount of well-connected data can be handled, as the expensive joint operation is unnecessary. Graph stores support transactions in the way that relational databases support them. The graph store allows the update in the database in an isolated location, hiding them from other processes until the transaction is executed.

Relationships are the priority in graph store database. Each entity contains a list of relationship records linked to other nodes. When performing a query to find those relationships, the database will use this list to find connecting nodes, reducing the time from searching and matching. Available graph databases include Neo4J, InfiniteGraph, and OrientDB. (Amirian, Basiri et al. 2014) state that geospatial data can be modelled as graphs. Graph stores support native topology, thus allowing the management of the relationship and connectivity

between geospatial data. The topological relationship has a major role in the geospatial data workflow. The graph store allows each edge to contain various properties, thus providing flexibility to navigate the network based on various properties such as, time, distance, number of points of interest based on user preferences in acquiring the best path and the unique path for every user.

4. UNSTRUCTURED DATABASE ANALYSIS

Unstructured databases represent an alternative to relational databases. Generally, they aim to tackle the drawbacks of relational databases in terms of schema rigidity, the limitations to horizontal and elastic scalability, and the support of specific data models. We analysed the storage environment for the 3D point cloud based on the three concepts above. Then, the unstructured database application is examined and compared to determine the capability for storing 3D point cloud data and which one has the best performance for database operations.

4.1 Data Model

The data model is a visual representation of the data and its connectivity. The data model for the unstructured database is different for every unstructured database category. Below, we explain the data model for four categories of the unstructured database.

4.1.1 Document Store

Document store data model using keys to locate documents inside the database. Most document stores represent documents using JSON (JavaScript Object Notation) or BSON (Binary JSON), which format is derived from it. The 3D point cloud contains massive data, which requires it to be stored in BSON format that encodes type and length information, allowing data to be analysed quicker. BSON support complex mathematics which is perfect for point cloud data from various sources that normally have different sized integers such as "ints" and "longs" or various decimal precision level such as "float" and "double" and "decimal". It is beneficial as it allows comparisons and calculations to happen in the data directly to distinct guise data stored that shorten consuming code.

In addition, the document stores are suitable for applications where the input data can be represented in a document format, such as a 3D point cloud. It can contain complex data structures such as nested objects and does not need adherence to a fixed schema. MongoDB is one of the popular applications for document storage in unstructured databases. It provides the extra functionality of grouping the documents into collections, in this case, a 3D point cloud. Inside each collection, the 3D point cloud document has a unique key unique to each coordinate point.

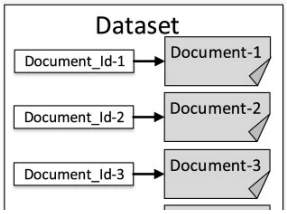

Figure 1. Document Store Database (Grolinger et al., 2013)

Figure 1 shows the data model for the document store database. Document store database store 3D point cloud in documents orientation, thus allowing each document to have a different structure. Each entity, specifically the 3D point cloud data, will be stored in one document, and documents are stored together in a collection. Each data in a document will have a unique identifier; Documet_Id-1, Documet_Id-2, Documet_Id-3, and Documet_Id-4. This is used to access documents in the collection. Document: Document-1, Document-2, Document-3, and Document-4 contain all the necessary information to describe the entity.

4.1.2 Key-Value

The data model for key-value stores is a simple model based on key-value pairs (Ramzan, Bajwa et al. 2019), which look like an associative map or a dictionary. The data model used the key with unique identifiers value to call the data in the database. The database value is obscure and is used to store data, such as an integer, a string, an array, or an object, thus delivering a schema-less data model. Besides being schema-free, key-value stores are very efficient in storing distributed data but are unsuitable for scenarios requiring relations or structures. All the functions needed relations or structures to be implemented in the application. Furthermore, the database cannot manage data-level querying and indexing because the values are obscure.

Figure 2 shows the data model for the key-value database. The value of each key can be anything from PDFs and JPEGs to JSON or XML documents, while the key itself specifies the element's URL. In this approach, the nodes in a cluster of machines can be used by the application designers to manage a lot of requests and online content.

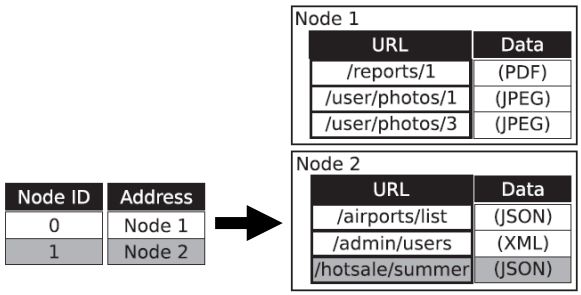

Figure 2. Key-value Database (Corbellini, Mateos et al. 2017)

4.1.3 Column Store

The data model for column stores is suitable for 3D point cloud data that deals with massive amounts of data stored on huge instances because the data model can be partitioned adeptly. The data model for the column store database is shown in Figure 3. The dataset consists of several rows referred to by a unique "Row-Key". Row-Key is known as a primary key. Each row is made of a set of "Column-Family", and each row can have a different "Column-Family". The 3D point cloud data is stored in "Column Name".

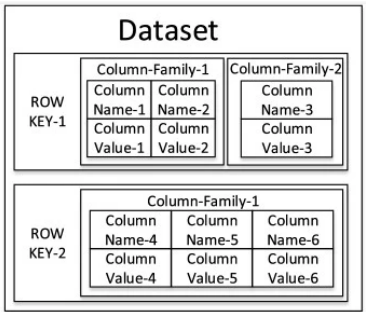

Figure 3. Column Store Database (Grolinger et al., 2013)

Makris, Tserpes et al. (2016) state that column stores can be visualised similarly to relational databases due to the table format. The major difference is handling the null value; a relational database stores a null value in each column while column stores only store a key-value pair in a row when the database needs it. In column stores, the column configuration is performed during the start. However, columns' prior definition is not required, which presents huge flexibility in inserting any 3D point cloud data type. Column stores offer more effective indexing and querying than key-value stores as they are based on column and row keys.

4.1.4 Graph Store

The data model in the graph store uses graphs as it originated from graph theory. Figure 4 shows the data model for the graph store database. The relationship in graph store is important as it relates two pieces of information. The data or the information are stored in "Key-Value Node", and the connections between Key-Value Node are called "Key-Value". The Key-Value is formed between each Key-Value Node, as shown in Figure 4.

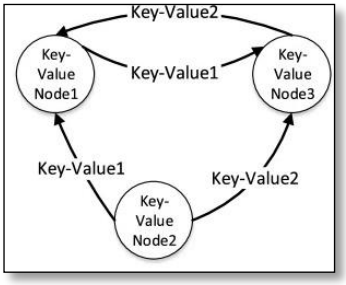

Figure 4. Graph Store Database (Grolinger et al., 2013)

The vertices, nodes, and edges that interconnect the vertices used the graph as a mathematical theory to represent a set of objects. Graph stores have a completely different data model than document, key-value, and column stores. Thus, it can efficiently store the links between other data nodes. In the graph store, the nodes and edges contain individual properties containing key-value pairs. It specialises in managing highly interconnected data and efficiently navigating relationships among different entities. This is unsuitable for 3D point cloud data as it does not focus on the relationship between the other entities. They are more appropriate for social networking applications, dependency analysis, and recommendation systems. Furthermore, the graph store is not as efficient as other unstructured databases in situations other than managing graphs and their relationships. Additionally, it is not efficient at horizontal scaling, reducing the performance as related nodes are stored on different servers, thus requiring navigating multiple servers.

4.2 Querying

Database querying capabilities are important when selecting a database based on a particular situation. Different databases offer different APIs and interfaces interaction. This is directly dependent on the data model that a specific database has. The important feature of an unstructured database query is MapReduce; it can perform distributed data processing on a cluster of computers. Thus, allowing unstructured databases to achieve one of the objectives: to scale over many computers. In addition, the REST-based API is a popular application because of it (Grolinger, Higashino et al. 2013). In the unstructured database, a REST-based interface offered a solution directly or indirectly through third-party APIs. We will explain the query methods for unstructured databases below.

4.2.1 Document Store

Document stores embed attribute metadata linked with values that the querying support range queries, indexing, and nested documents. Also, it allows the use of operations like OR, AND, BETWEEN, and queries that can be executed as MapReduce. For example, MongoDB supports creating, inserting, reading, updating, and removing operators. In addition to supporting manual indexing, indexing on embedded documents and indexing location-based data (Makris, Tserpes et al. 2016). The queries are flexible, as there is no need to stick to a predefined schema to store data. MongoDB supports standard GeoJSON data types for 3D point cloud data, including Point, Polygon, MultiPolygon, and GeometryCollection. It can implement a subset of basic spatial operations using 3D point cloud data for inclusion, intersection, and proximity.

4.2.2 Key-Value

The query operations for the database are linked with a key. It is accessible through APIs, including functions like GET, PUT and DELETE. However, the interfaces only facilitate simple queries to be executed as complex queries are not supported, rendering the query language unnecessary (Makris, Tserpes et al. 2016). Thus, reducing the chances of 3D point cloud handling as it does not support complex spatial queries such as intersection and proximity. The key value can be used to store 3D point cloud data, but the complexity of geospatial data obstructs spatial searches, particularly for polylines and

polygons. Consequently, spatial indexed is required for fast data retrieval, affecting performance.

4.2.3 Column Store

Column stores query using a regular expression on the indexed values or the row keys. The impact on the databases is the operations are only affecting the query related rather than the entire row from the storage (Makris, Tserpes et al. 2016). Column stores use distributed environments to store data storage, support horizontal scalability, and are designed to run on many devices. However, the queries are limited to keys and usually do not execute the query by column or value. Nevertheless, limiting queries to just keys ensures faster locating the device containing actual data. Since it includes limited transaction support, there are no joint operations in the column store as in other unstructured databases. Column stores are ideal for storing huge 3D point cloud data when high availability is required. Like document store, there are many columns store that supports geospatial data management and simple analysis. 3D point cloud data handling requires heavy data insertion and quick data retrieval efficiently uses column stores with simple queries.

4.2.4 Graph Store

The graph store is suitable for applications based on data with many relationships because expensive operations like joins are substituted by efficient navigation such as graph and graph pattern matching technique. In the graph store, the query processing begins from one node and then travels to the other nodes per the query description. They define pattern locate graph pattern matching technique in the original graph. Makris, Tserpes et al. (2016) give the instance for Neo4j, where each vertex and edge in the graph store a "mini-index" of the connected objects. This implies that the graph size does not affect the performance upon a traversal and the local hop cost remains the same. The global adjacency index is only used when trying to locate a traversal's beginning point. Indexes must promptly retrieve vertices based on their values and provide a traversal beginning point.

4.3 Scaling

Unstructured databases are designed for horizontal scaling. The data is distributed, which increases database capacity by adding nodes to the database. Sharding is often used to achieve horizontal scaling, which involves splitting the data records into several independent partitions or shards using a given condition. Unstructured databases do not support the join function; thus, the developer can decide whether to perform joins at the application level, which may involve gathering data from several physical nodes based on some conditions and then joining or combining the collected denormalise data. This requires more development effort, but frameworks such as MapReduce can considerably lessen the task by providing a programming model for distributed and parallel processing (R 2020). There are four methods of scaling: partition, replication, consistency, and concurrency.

4.3.1 Partitioning

In the context of big geospatial data, the volume of information used to store 3D point cloud data is challenging. For a single-node system, increasing the storage capacity of any computational node means adding more RAM or disk space under the constraints of the underlying hardware. Once a node reaches its storage limit, there is no alternative but to distribute the data among different nodes. The data partition is a common method to store and process massive 3D point clouds, and the partitions are stored across various server nodes. The unstructured database implements high availability and scalability solutions leveraged in cloud environments. Partitioning means that every instance will only hold a subset of keys. Two main partition ways are range, hash partitioning, and consistent hashing. Previous study by Azri et. al. (2014) and Azri et. Al. (2016) successfully shows 3D points are portioned using segmentation and clustering approaches. The study described how 3D points can be group into several partions for efficient data retrieval.

MongoDB, a document store database, supports horizontal scalability, and the queries are distributed by exploiting the sharding. MongoDB partition supports geospatial attributes as sharding attributes. It is accomplished through sharding, either manual or automatic. In manual sharding 3D point, cloud data are set up to two MongoDB main servers. Half of the 3D point cloud data is stored on one and the remainder of point cloud data on the other. The application component takes care of all the 3D point cloud data splitting and recombination in auto-sharding. For efficient point cloud retrieval and queries, the application will ensure that the data is inserted into the right server. The 3D point cloud data collection will be stored in chunks, and each chunk will be stored on a different server. Each server executes the query on its data chunk when executing a query, thus parallelising the execution. The data partition is based on the value of the selected geospatial sharding attribute. Hence, the choice of the geospatial sharding attribute is crucial for the server to attain a balanced data distribution. The geospatial sharding attribute is selected based on the predicates of the expected queries that are frequently used (Baralis, Dalla Valle et al. 2017).

In Dynamo the key-value database, the partition depends on consistent hashing. The output of consistent hashing is called a "ring", and each node is given a random value in the ring. A key identifies each 3D point cloud data given to a node by hashing the point cloud key to generate its position on the ring, then moving the ring clockwise to find the first node with a position larger than the point cloud data item's position. Hence, it becomes each node's responsibility for the region in the ring, between the ring and the predecessor node of the ring. To ensure data consistency and avoid redundancy, each node is mapped to multiple points in the ring instead of a single point, called virtual nodes. Every virtual node seems like a single node in the network, but each node will oversee more than one virtual node. In Cassandra, a column store database, the data partition across the instances is achieved through consistent hashing using an order-preserving hash function. The consistent algorithm that is used is the same as in the case of Dynamo without virtual nodes. For the graph store database,

Neo4j, the data partition is direct reads to instances where the database will already have those 3D point clouds in memory. This approach is significantly beneficial when the active 3D point cloud data is larger than its capability to fit into memory in any instance.

4.3.2 Replication

Replication is an important operation related to an unstructured database's dependability and data consistency. It is the process that stores the same data on several servers, thus distributing the read and write operations. MongoDB uses main-sub replication, which indicates that only one database is active for inserting at any given time. By passing the data into the main database and to a replica, the sub-database of the main database, there is no need to worry if the main database fails as the sub-database can substitute it. In Dynamo, data replication is used to provide availability and durability. Each 3D point cloud data is replicated for N-times where N can be configured "per instance". The node will oversee the data with key-value k and store the updated replicas in N-1 nodes. For each k, there will be a reference list containing nodes where the data item with k must be stored. Each data for Cassandra is duplicated over N hosts, where N is the defined "per-instance" replication factor". Each k is assigned to a coordinator node. The coordinator oversees the replication of the data items within its range. The replication in Neo4j follows a main – sub architecture, where all insert is committed from a single main instance. Thus, the sub will receive it before persevered to other cluster instances.

4.3.3 Consistency

Consistency ensures that a transaction brings into the database from one valid state to another. The key consistency models are strong and eventual consistency. Strong consistency ensures that the updated 3D point cloud data are visible to all following read requests when write requests are verified. However, although synchronous ensures strong consistency, it also creates latency. Asynchronous replication led to subsequent consistency as there is a lag between write verification and transmission. In the eventual consistency model, alterations eventually reproduce through the application for sufficient time. Hence, some server nodes may contain inconsistent data for a period.

MongoDB supports immediate consistency, which means the application limitations of updates to a single main node for a given data part. All the updates are made in place. Thus, MongoDB allows chances in 3D point cloud data update wherever it happens. Hence, all the changes in the 3D point cloud are displayed simultaneously in all database servers. In Dynamo, consistency is assisted by object versioning. A quorum-like technique and a decentralised replica synchronisation protocol support the consistency among replicas during updates. In Cassandra instances a read and write request for a key is transmitted to any node. For writing, the system routes the requests to the replicas and waits for a quorum of replicas to acknowledge the end of the writing. While for reading, the system either transmits the requests to the closest replica or all replicas and then waits for a quorum of responses based on the consistency securities required by the client. Hence, it takes longer for the 3D point cloud to display. Neo4j supports eventual consistency, where all updates eventually transmit from the main database to another sub-database, so an insert from one sub-database may not be immediately visible on another sub-database. Hence, changes on the 3D point cloud in one server will slowly affect the other server.

4.3.4 Concurrency

All the database environments are no omission to hardware failures, including unstructured databases. Nevertheless, the unstructured database's distributed architecture ensures no single point of failure and the built-in redundancy for both data and function. If more than one database server or node collapses, the other nodes in the system can resume the tasks without data loss, thus demonstrating true fault tolerance. In this way, the unstructured database can provide consistent uninterrupted 3D point cloud data for the user, whether in a single location, throughout the data centres, or in the cloud.

5. DISCUSSION

Table 1: Overview of Unstructured Database

Unstructured Database	Example	Data model	Geospatial Query	Scaling		
				Portioning	Replication	Consistency
Document-store	MongoDB	Document like structure with flexible schema and no predefined model	Supports query for Point, LineString, Polygon, MultiPolygon,MultiLineString, and GeometryCollection.	Range portioning based on a shard key	Main-sub, asynchronous replication	Immediate
Key-value	Dynamo	Simple model based on key-value pairs which look like an associative map	Supports query for Point	Consistent hashing	Synchronous/ asynchronous	Eventual

Column-store	Cassandra	Table format with a key-value pair stored in a row	Supports query for Point, LineString. And Polygon	Consistent hashing and range partitioning	Masterless, asynchronous replication	Based on read and write requests
Graph-store	Neo4j	The vertices, nodes, and edges interconnect the vertices	Neo4j spatial library support Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon and Geometry Collection	Cache-based	Master-slave	Eventual

The advantages of using unstructured databases in 3D smart cities can be clarified using the geospatial big data challenges of volume, variety, and velocity. The massive 3D point cloud data is collected from high mobile devices, spatial-temporal resolution satellites, and airborne photos. GPS locations (Global Positioning System) devices are used to visualise 3D smart city. A highly scalable storage solution is needed for the massive volume 3D smart city model, which is easy for unstructured databases as it allows horizontal scaling across the distributed database. An unstructured database supports various geospatial data formats from multiple sources, making it a flexible database. It can store complex data types that contain spatial elements such as geotagged photos or videos and timelines associated with places from social media. This information falls into unstructured or semi-structured data and can be stored in an unstructured database. 3D point cloud processing and analysis require a short timeframe, described as a velocity demand. 3D point cloud needs fast processing and querying for predictive algorithms to be utilised in real-time streaming data. In addition, to develop a 3D model, the hardware and software selection is important (Ariff, Azri et al. 2020) thus, selecting the right database and the software for the database is important. The unstructured database is a splendid solution to the geospatial big data challenges.

6. CONCLUSION

In 3D point cloud handling, each database has certain crucial characteristics designed to achieve high-performance data retrieval, availability, and scalability. However, each unstructured database has a variety in terms of the data model, database interfaces, and features. In this paper, we reviewed the four categories of the unstructured database: document store, key-value, column store, and graph store, and the difference between them in terms of 3D point cloud handling, data model, querying, and scaling. Based on the systematic comparison, we conclude that a document store is the most suitable storage environment for a 3D point cloud from four major categories of the unstructured database. This paper's contribution assists the developer in understanding the differences between all categories of an unstructured database.

ACKNOWLEDGEMENT

This work was supported by Ministry of Education (MOE) Malaysia, through Fundamental Research Grant Scheme (FRGS/1/2022/WAB07/UTM/02/3).

REFERENCES

Amirian, P., et al. (2014). Evaluation of Data Management Systems for Geospatial Big Data. Computational Science and Its Applications – ICCSA 2014, Cham, Springer International Publishing.

Ariff, S. A. M., et al. (2020). "Exploratory Study of 3d Point Cloud Triangulation for Smart City Modelling and Visualization." The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLIV-4/W3-2020: 71-79.

Atzeni, P., et al. (2020). "Data modeling in the NoSQL world." Computer Standards & Interfaces 67.

Azri, S., Ujang, U., Abdul Rahman, A., 2020. Voronoi classified and clustered data constellation: A new 3D data structure for geomarketing strategies. ISPRS Journal of Photogrammetry and Remote Sensing 162, 1-16

Azri, S., Ujang, U., Rahman, A.A., Anton, F., Mioc, D., 2016. 3D Geomarketing Segmentation: A Higher Spatial Dimension Planning Perspective. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XLII-4/W1, 1-7.

Azri, S., Ujang, U., Anton, F., Mioc, D., Rahman, A.A., 2014. Spatial Access Method for Urban Geospatial Database Management: An Efficient Approach of 3D Vector Data Clustering Technique, 9th International Conference on Digital Information Management (ICDIM). IEEE, Bangkok, Thailand.

Azri, S., et al. (2021). "Document-Oriented Data Organization for Unmanned Aerial Vehicle Outputs." Journal of Advances in Information Technology 12(4).

Baralis, E., et al. (2017). SQL versus NoSQL databases for geospatial applications. 2017 IEEE International Conference on Big Data (Big Data): 3388-3397.

Cattell, R. G. G. (2011). "Scalable SQL and NoSQL data stores." SIGMOD Rec. 39: 12-27.

Corbellini, A., et al. (2017). "Persisting big-data: The NoSQL landscape." Information Systems 63: 1-23.

de la Vega, A., et al. (2020). "Mortadelo: Automatic generation of NoSQL stores from platform-independent data models." Future Generation Computer Systems 105: 455-474.

Grolinger, K., et al. (2013). "Data management in cloud environments: NoSQL and NewSQL data stores." Journal of

Cloud Computing: Advances, Systems and Applications 2(1): 22.

Li, Z. (2018). "NoSQL Databases." Geographic Information Science & Technology Body of Knowledge 2018(Q2).

Liu, K. and J. Boehm (2015). "Classification of Big Point Cloud Data Using Cloud Computing." The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XL-3/W3: 553-557.

Luan, H., et al. (2014). Towards Effective 3D Model Management on Hadoop: 131-139.

Mohd, Z.H., Ujang, U., 2016. Integrating Multiple Criteria Evaluation and GIS In Ecotourism: A Review. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. XLII-4/W1, 351-354.

Makris, A., et al. (2016). "A Classification of NoSQL Data Stores Based on Key Design Characteristics." Procedia Computer Science 97: 94-103.

Mao, B., et al. (2014). "NoSQL Based 3D City Model Management System." The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XL-4: 169-173.

Mior, M. J., et al. (2016). NoSE: Schema design for NoSQL applications. 2016 IEEE 32nd International Conference on Data Engineering (ICDE).

Park, J.-W., et al. (2011). Visualization of Urban Air Pollution with Cloud Computing.

R, B. (2020). "Map Reduce: Data Processing on large clusters, Applications and Implementations." 05: 214-220.

Ramzan, et al. (2019). "Challenges in NoSQL-Based Distributed Data Storage: A Systematic Literature Review." Electronics 8(5).

Reniers, V., et al. (2019). "Object to NoSQL Database Mappers (ONDM): A systematic survey and comparison of frameworks." Information Systems 85: 1-20.

Salleh, S., Ujang, U., Azri, S., 2021. Virtual 3D Campus for Universiti Teknologi Malaysia (UTM). ISPRS International Journal of Geo-Information 10.

Višnjevac, N., et al. (2019). "Prototype of the 3D Cadastral System Based on a NoSQL Database and a JavaScript Visualization Application." ISPRS International Journal of Geo-Information 8(5).

Wan Abdul Basir, W.N.F., Majid, Z., Ujang, U., Chong, A., 2018. Integration of GIS and BIM techniques in construction project management - A review, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives, 4/W9 ed, pp. 307-316.

Xu, X. and R. Guo (2016). Research on Storage and Processing of MongoDB for Laser Point Cloud under Distribution.

Zhang, D., et al. (2019). "Improving NoSQL Storage Schema Based on Z-Curve for Spatial Vector Data." IEEE Access 7: 78817-78829.