Nina Nguyen

CPSC 5890

Fall 2019

**Big Data Scalable Architecture**

1. **Abstract**

Living in the information age, data is currency. It is estimated that by 2025, 463 exabytes of data will be generated each day globally. With the rise of numerous information gathering devices, the data being gathered are too complex and too large for traditional databases to store and analyze. Big data is a group of large data sets that include complex data. This paper analyzes the challenges of storing and processing big data, then explore the different methods of storing and processing of big data, such as MapReduce and NoSQL databases. Furthermore, this paper delves into an open scalable model that incorporate both traditional methods as well as NoSQL in order to process and store big data efficiently.

1. **Introduction**

Data is backbone of any business. From organizational to analytical, storing and processing the data can make or break an enterprise. Organizations usually store their data in relational databases. However, a lot of data being generated today are from blogs, email, images, videos, social media, etc. These types of data are unstructured data and mining these kinds of data is becoming extremely useful. Non-traditional data accounts for 80% of enterprise data and continue to rise, this create focus on bringing traditional and non-traditional data together under a common umbrella [4].

The traditional method in storing and handling data is through relational databases, which date back to 1970s. A typical database schema design can be defined by a conceptual data model, mapping it to a relation data model, normalize the relations, and apply various optimizations such as tables and indexes.

Nowadays, the amount of data that are available increased and being generated at an extreme rate making the requirements needed to handle these data different than they were in the past. Relational database management systems while great at handling structured data, often have trouble managing big data. Big data refers to large and complex datasets and can be categories into four categories: Volume, Velocity, Variety, and Veracity [1].

Volume measures the quantity of data. Some small organizations may have terabytes or gigabytes of data, but large organizations could reach petabytes or exabytes of data. Velocity refers to the continuously streams of data that are being generated and processed. Variety captures different type of data, it can be structured, semi-structured or unstructured data. Finally, veracity attribute to the accuracy of data. The data can be useless and problematic if it’s not accurate.

There are three different data types in big data: structured, semi-structured, and unstructured [1]. Structured data adheres to a pre-defined data model with relationship defined between the rows and columns, so it is easy to analyze. Semi-structured data does not conform with a formal data model but does contain tags or other markers to enforce some hierarchies of records and fields within the data. On the other hand, unstructured data does not adhere to any data model causing a lot of irregularities and ambiguities.

Most of the data that are stored in today’s digital world are unstructured data, traditional SQL databases cannot process these types of data in an efficient way. Therefore, NoSQL emerges as a popular data store for big data. NoSQL (non-relational databases) are highly distributed databases that focuses on scalability. There are two types of scaling; vertical scaling or scaling-up, and horizontal scaling or scaling-out. Vertical scaling means increasing hardware devices to change an existing machine, while horizontal scaling means adding more machines.

Two different principles were derived from the CAP theorem on SQL databases and NoSQL databases. The first one is ACID (**A**tomicity, **C**onsistency, **I**solation and **D**urability) that focuses on consistency and availability. The second principle is BASE (**B**asically **A**vailable, **S**oft state, and **E**ventual consistency) where partition tolerance and availability are preferred over consistency [3]. NoSQL databases adopts BASE and can be classified into four main types of database; key-value, document, wide-column, and graph databases. There are other NoSQL databases, but they won’t be covered in this literature review.

While NoSQL provides a great and efficient way to store big data, Hadoop MapReduce and Apache Spark have proven to be strong programming paradigms for processing large data sets in distributed environments [1].

The rest of the paper is organized as follows. Section 3 and 4 explore MapReduce and Apache Spark’s role in processing data as part of literature review, Section 5 continues the literature review and details multiple powerful NoSQL stores, Section 6 analyzes the proposed Open Scalable Relational Data Model, section 7 concludes the report and section 8 list the references.

1. **Literature Review – MapReduce [1]**

MapReduce is a scalable programming model capable of processing big data sets using a parallel, distributed algorithm on commodity computing nodes. MapReduce divide and conquer a problem using the “map” phase and the “reduce” phase. The map function handles filtering and sorting while the reduce function groupings and aggregate.

The flow of MapReduce consists of a single master node and multiple worker nodes. The master node breaks down the input into multiple pieces called split or shard, and then assign them to map workers [1]. The workers then process the inputs and generate corresponding key/value pairs. The master then notifies the reduce workers and the key/value pairs are then processed and outputted to files.

Some of the advantages of MapReduce are Scalability, Flexibility, and Parallel processing. Large data sets can be stored across multiple servers, and servers can operate in parallel. By adding more servers, the processing power increases. Traditional relational database management systems are not able to horizontally scale and process large data sets. While MapReduce can handle large data sets as well as offers flexibility to process structured, semi-structured, and unstructured data. It can process the data by dividing the task in a manner that allows the execution of the independent task in parallel. This improves the time it takes to process a task.

Although MapReduce have proven to be a dominant algorithm to handle big data processes, there are some limitations on what it can do. First, it is not always easy to implement each and everything as MapReduce. The algorithm is complex and incorporating that into an existing database can be tricky, this is known as interoperability. Vendors in Hadoop and NoSQL are developing adapters to solve these integration problems [4]. MapReduce also lacks a high-level language such as SQL so data exploration can be difficult. Other than complexity, MapReduce is best suited for batch processes so it can be a challenge when it comes to stream processing of data. Users can’t expect MapReduce to be fast, but it is definitely powerful.

1. **Literature Review – Apache Spark [7]**

To make up for the drawbacks of MapReduce, Apache Spark was designed to function in a Hadoop environment and work as an alternative to MapReduce. Although, it can be run with or without Hadoop. Spark is an open source processing engine that mainly focuses on speed and reliability. Unlike MapReduce, where it has to read and write data from disk causing performance to suffer, Spark copies the data into RAM reducing the time it needs to interact with physical servers. This makes Spark 100x faster in memory and 10x faster on disk. To handle failures, Spark uses Resilient Distributed Dataset (RDD), an immutable distributed collection of objects.

Depending on the businesses’ needs, Hadoop MapReduce is slower but works great for batch processing of large datasets, especially structured data. Apache Spark can handle real-time processing and Machine Learning as well as being the best solution when analyzing any type of unstructured data.

1. **Literature Review – NoSQL [3]**

NoSQL databases emerges as a response to relational data models limitation. A popular design choice from NoSQL is “shared-nothing” architecture. This design means each server node in a cluster works independently from other nodes. No consensus is needed from other nodes in order to answer queries, so this improve queries latency.

There are different approaches to NoSQL databases, but what they have in common is that they are not relational [8]. As mentioned, the four main types of NoSQL systems; key-value, document, wide-column, and graph databases [6]:

1. Key-Value stores – Each value is assigned a unique key. Key-value is great for caching, queue, and live updates of information. This is the simplest NoSQL databases; some examples are Riak and Berkeley DB.
2. Document stores – Similar to key-value but can only store documents like JSON or XML as values. This database has some drawbacks, one being productivity. For example, if one field is needed from the document, the whole document is accessed during the query. Some examples of document store are MongoDB and CouchDB.
3. Wide-column store - Uses columns to store data. It offers high performance and scalable architecture. Some examples of wide-column stores are Cassandra and HBase.
4. Graph store – The basis of graph databases includes nodes and relationships. Node represents an entity/data and relationship represents how two nodes are related. An example of graph store is Neo4J.

A popular NoSQL wide-column database is Apache Cassandra. It is linearly scalable, distributed, and fault-tolerant on a peer-to-peer architecture. Cassandra provides a declarative and user-friendly Cassandra Query Language (CQL) for easy access and processing of the data. There is no single point of failure in Cassandra, all nodes are created equal in a cluster. Data is then distributed across all nodes and each node is capable of handling read and write requests.

Since Cassandra is designed to achieve superior write and read performance, data modeling for this database should start with application queries. Schema denormalization is very common in Cassandra in order to enable complex queries to be answered by only accessing a single table [5]. This also leads to data duplication because the data might be in another table to support another query since there is no join. However, disk space is the cheapest resource and Cassandra is architected around that fact.

All the attributes above allow Cassandra to manage one of the largest datasets on clusters with thousands of nodes distributed across multiple data centers.

1. **Open Scalable Relational Data Model [2]**

When integrating traditional two-dimensional tables to distributed systems, there are complications configurations of database and table division. This make it difficult to partition data-item-based data and limit the scalability performance of distributed environment. This led researchers to try two approaches. In the first approach, they focused on enhancing and broadening existing relational data models to meet the requirements of big data applications. This included a column storage strategy where query performance is excellent but incurs significant space overhead and does not solve the scalability limitation that is pervasive in relational data models. In the second approach, the researchers focused on using NoSQL data models the address the performance and scalability when accessing data. However, the data’s descriptive power that traditional models provide will be lost and reduce functionality.

In order to address both limitations of traditional models and NoSQL models, the researchers proposed an Open Scalable Relational Data Model (OSRDM) that has data support and enable complete horizontal scalability. It does so while maintaining descriptive powers of relational data models.

OSRDM also support user-defined features and this opens up flexibility of the model. Users can design the architecture to their business needs.

1. **Data Structure**

OSRDM data structure can be described as a data set D (MK|FK|T|V|R). MK is a collection of main keys, where each key identifies specific data items similar to how key-value store works. Keys can then be hashed or sectioned in order to achieve horizontal scalability. FK stands for featured keys, where the description of features for data items are held. The description can be preset system values or user-defined and held in fixed-length sections. T defines the data types, it can be system preset or user-defined. A user-defined data structure is defined by data types through structs. For example, to define a “produce” struct can be described as:

*Define Struct produce{*

*int weight;*

*double price;*

*}*

After defining the data structure, the getDes() function can be called to get the description of the data structure. V holds the actual value of the data items not including the metadata. And finally, R describe the relationship between data items. It can be defined by R(FromItem, ToItem, Type, Flag). Relationships can also be user-defined as:

*Define relationType(des, type)*

1. **Data Operations**

OSRDM is similar to traditional relational models by supporting operations such as: select, insert, and insert relations. It is similar to NoSQL by supporting sort, aggregate and generate graph.

1. Selecting data items from “dataSet1” where MKey equal a certain value1 can called as follows:
2. *Select items from dataSet1 [where [MKey equals value1]]*
3. Inserting data into the database can be done by calling:
4. *InsertData(mainKey, datatype, data, [keyTag1], [keyTag2],…,[keyTagN])*
5. There are multiple ways relations can be inserted into a database. It can be inserted item-to-item, item-to-set, set-to-item or set-to-set relation. An example on how to run this query is as follows:
6. InsertRelationOneToOne(type, fromItem, toItem)
7. This Open Scalable model incorporated both traditional models and NoSQL models features in their sort data operation. NoSQL models generally does not support indexing so by using the power of NoSQL and performance of traditional models, sorting can be done in Open Scalable model. Sorting dataset1 by featured keys with rule1 then generating index “index1” can be done by calling:
8. *Sort dataset1 to index1 by Fkey with rule1.*
9. Similar to sorting, aggregating data items can be done with a rule and then index will be generated for the results. To aggregate dataset1 by applying rule1 to the Fkey, it can be done using the query:
10. *Aggregate dataSet1 to index1 by [Fkey with rule1]*
11. Generate Relational Graph, which is cannot be done in traditional models allows for semi-structured or unstructured data to be represented. Data can be returned in two forms, two-dimensional tables or using relational graphs in two-dimensional or even three-dimensional forms. To generate a graph, the query is:
12. *Generate graph dataSet1*

The researchers designed OSRDM to use an open method of data description to work efficiently with any type of data, this sets it apart from traditional relational models and NoSQL models. The user can use system preset data types or define their own to meet their business needs. The flexibility really set this model apart and allow for a sandbox type of environment, which is not very common. In addition to that, traditional models use two-dimensional tables to represent relations, OSRDM uses linked-data form to perform the same feat. This allows for dataset to be understandable by humans and machines alike while making OSRDM so powerful.

Popular NoSQL models such as key-value models and column-oriented models has no representations of relations between the data, everything is group into a single data item. This is how horizontal scalability is achieved. On the other hand, traditional models use two-dimensional tables and foreign keys to define relationships which makes horizontal scalability difficult. OSRDM model workaround for that limitation is separate relations from structures and giving them the same status as data items. This just means using traditional relations as a guide and alternating the way relations are viewed in NoSQL. This allows for OSRDM to achieve full horizontal scalability equal to NoSQL models.

Overall, OSRDM takes the best of traditional models and NoSQL models to create a hybrid model. With the trend that Big Data is moving toward, OSRDM might be the solution to a lot of problems. There are still a lot of optimizing to do and functionalities to add, and security guardrails to be implemented. However, in the future, the authors plan on releasing this database product to provide actual practical use.

1. **Conclusion**

With the amount of data that are being generated every day, the traditional relational databases can no longer store and process the data efficiently. With the use of MapReduce, large amount of data can be processed at parallel using commodity computing nodes. Apache Spark can pick up where MapReduce left off by processing data in micro-batches copying data to RAM instead of disks, this allows lower the latency significantly and allow for real-time stream of data. Different NoSQL databases can store different types of Big Data depending on the business’ needs. Open Scalable Relation Data Model is a proposed model that combine both features from traditional databases as well as NoSQL database to create a fast and powerful data store. Even in its early stages, the model performance is similar to MySQL and HBASE when running random queries. OSRDM does outperform MySQL when it comes to more complex queries. There are a lot of optimization to be done and new features added to the model. By then, we will actually see how well the model perform when it comes to Big Data.

The future of Big Data technologies is very promising even at its infancy. Traditional models have a lot of matured systems, so development scalability and maintainability are not a major concern. However, Big Data technology is still new especially when not a lot of enterprises have adopted the use Big Data. There are a lack of IDEs, Testing, Deployment and Administration tools which can make adopting Big Data a pain [4]. However, like Open Scalable Relational Data Model, there are going to be more technologies that make Big Data easier to use.

The bottom line is that currently, there is no single datastore or programming paradigm that can totally handle or process big data completely. There are always some give or take with each technology. However, they have proven to be better than traditional models when it comes to Big Data, and hybrid models like Open Scalable Relational Data Model might be more common in the future as more enterprises start to incorporate Big Data into their business.

1. **References**

[1] K. Grolinger, M. Hayes, W. A. Higashino, A. L'Heureux, D. S. Allison and M. A. M. Capretz, "Challenges for MapReduce in Big Data," *2014 IEEE World Congress on Services*, Anchorage, AK, 2014, pp. 182-189.

[2] Z. Zheng, Z. Du, L. Li and Y. Guo, "BigData Oriented Open Scalable Relational Data Model," *2014 IEEE International Congress on Big Data*, Anchorage, AK, 2014, pp. 398-405.

[3] A. Hendawi *et al*., "Distributed NoSQL Data Stores: Performance Analysis and a Case Study," *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 1937-1944.

[4] S. Dhar and S. Mazumdar, "Challenges and best practices for enterprise adoption of Big Data technologies," *2014 IEEE International Technology Management Conference*, Chicago, IL, 2014, pp. 1-4.

[5] A. Chebotko, A. Kashlev and S. Lu, "A Big Data Modeling Methodology for Apache Cassandra," *2015 IEEE International Congress on Big Data*, New York, NY, 2015, pp. 238-245.

[6] A. Kuzochkina, M. Shirokopetleva and Z. Dudar, "Analyzing and Comparison of NoSQL DBMS," *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, Kharkiv, Ukraine, 2018, pp. 560-564

[7] K. Aziz, D. Zaidouni and M. Bellafkih, "Real-time data analysis using Spark and Hadoop," *2018 4th International Conference on Optimization and Applications (ICOA)*, Mohammedia, 2018, pp. 1-6.

[8] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," in *Computer*, vol. 43, no. 2, pp. 12-14, Feb. 2010.