ISSN: 0975-833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 7, Issue, 04, pp.14697-14701, April, 2015

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

RESEARCH ARTICLE

STRUCTURING UNSTRUCTURED BIG DATA

***Dr. Smitha Rao, M. S. and Kohila Kanagalakshmi, T.**

Department of MCA DSCASC, Bangalore, India

ARTICLE INFO

Article History:
Received 27th January, 2015
Received in revised form
20th February, 2015
Accepted 14th March, 2015
Published online 28th April, 2015

Key words:

Big Data, NoSQL,
Data Wrangling,
BASE,
CAP.

ABSTRACT

Relational databases which use SQL have a long- standing position in most organizations and have dominated the database markets for a long time. Relational databases work best with structured data—such as a set of sales figures—which readily fits in well-organized tables. This is not the case with unstructured data, such as that found in word-processing docu- ments, social media, log files etc. Especially in large scale and high concurrency applications, such as search engines and Social media using the relational database to store and query dynamic user data has been found to be inadequate. The advent of Big Data created a need for out-of-the-box horizontal scalability for data management systems. This ushered in an array of choices for Big Data Management under the umbrella term NoSQL, new technology for storage of large unstructured data. NoSQL databases are today gaining importance due to their linear scalability, schema flexibility and comparatively higher performance This paper highlights the implementation of NoSQL databases. It seeks to analyze various solutions to structure the unstructured data in terms of the way data is stored and retrieved from NoSQL databases.

INTRODUCTION

A relational database is a set of structures containing data fitted into predefined categories (Leavitt, 2010) known as tables that are organized as row and columns. Users can access data in different ways using joins without having to reorganize the database tables. Over 2.5 Exabyte of data is generated every day. This data can be accumulated from various sources like transactions from a large stock exchange captures more than 1 TB of data every day, data transmitted from about 1.75 billion smart phones, more than 48 hours of video every minute from YouTube, data from social media such as Twitter and Facebook captures more than 10 TB of data daily etc.With rapid growth of the Internet, product related word-of-mouth conversations have migrated to online markets, creating active electronic communities that provide a wealth of information. Millions of users share their opinions on social media platforms like Twitter, Facebook etc. making them a valuable platform for tracking and analyzing public sentiment. Such tracking and analysis can provide critical information for decision making in various domains. But the data from these sources are not necessarily structured. The major types of big data can be primarily classified as Structured data - which can be represented in a well designed and rigidly defined tabular format, Semi- Structured data - data which does not have a formal data model like XML-data and Unstructured - data which does not have a pre-defined\data model like the data found in log files, blogs, mails etc. Apart from the variation in the structure of the data the four main parameters characterizing Big Data include volume, velocity, variety and veracity. Due to these concerns storage and retrieval of semi-structured and unstructured data poses serious issues. NoSQL databases are today gaining importance due to their linear scalability, Schema flexibility and comparatively higher performance. Few of the prominent NoSQL databases are CouchDB, Cassendra, Neo4j, mongoDB etc. This paper aims at analyzing various solutions to design storage mechanisms for unstructured data to optimize their performance in terms of analysis and performance.

The organization of the paper is as follows. The concept and classification of Big Data are presented in section II, the emergence of NoSQL databases are discussed in Section III. Section IV presents the concepts of structuring unstructured big data in prominent NoSQL databases and finally conclusions are drawn in section V.

Big data

Data that is too big and complex to capture, store, process, analyze, and interpret using the state-of-the art tools and methods is referred to as Big Data. According to IBM, 90% of the data in the world today has been created in the last two years alone. This data is accumulated through components like climate information through sensors, data accumulated though social media, transaction information through financial

Corresponding author: Dr. Smitha Rao, M. S.
Department of MCA DSCASC, Bangalore, India.

institutions, online marketing components, data from cell phones etc. The four dimensions of big data are Volume, Variety, Velocity and Veracity. Volume refers to how much data has to be stored and processed, Velocity refers to the speed of generating and processing data, Variety focuses on the type of data that is to be analyzed and Veracity refers to the accuracy of the data in predicting business worth. Prior to processing the data it needs to be verified in terms of both accuracy and context. The real issues concerning big data is not the accumulation of data but the analytics to be performed to gain insights from the massive amounts of data. The insights could be for reducing costs and time, developing new products, finding the root cause of failures, optimizing business performance through smarter and in time decision making etc. A Big Data project is normally typified by the data variety – storage of data that is structured, semi-structured and unstructured. The phrase "structured data" usually refers to information that reside in a traditional row-column database. In semi-structured data there is no separation between the data and the schema and the amount of structure used depends on the purpose. Example XML documents, logs and EDI are all forms of semi-structured data. But "unstructured data" often include text and multimedia content. Examples include e-mail messages, word processing documents, videos, photos, audio files, presentations, webpages and many other kinds of business documents and they doesn't fit neatly in a database. Better data structures to represent data and efficient data stores to store and efficiently retrieve data from are the need of the day. File systems and Relational databases were found to be inadequate. Data must be viewed holistically. The emergence of NoSQL databases has proved to fruitful solving many of the Big Data related issues. Apart from tackling big data issues modern databases should also assist in tall the phases of information cycle – Acquire, Store, Process and Use/Present.

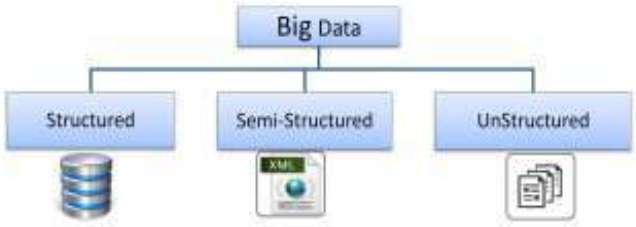

Fig.1. Classification of Big Data

NoSQL databases

Relational Database Model developed by mathematician Edgar F Codd is an hierarchical file system that allows records to be accessed using simple index system. Today RDBMS is insufficient to maintain the large volumes of data generated. This deficiency has given rise to an alternative system - NoSQL i.e. Not Only SQL that encompasses a wide range of database products that relies on various flexible data models. The main features of NoSQL databases are:

- Dynamic Schema
- Linear Scalability
- Throughput Oriented Architecture
- Continuous Data Availability
- Running well on clusters
- Mostly open-source
- Eventually consistent/BASE

A NoSQL application works with data-models that are specifically designed and optimized for applications. NoSQL databases allows developers to develop without having to convert in-memory structures to relational structures. In 2000, Eric Brewer popularized the acronyms "BASE"(Basically Available, Soft state, and Eventual consistency) and "CAP" (Consistency, Availability and Partition Tolerance) in his talk at ACM Symposium. Real time decision support system requiring big data analytics can be eventually consistent rather than rigidly following ACID properties of relational databases. The CAP theory mentioned by Brewer explains the theoretical divide between ACID and BASE compliant databases. Understanding the requirements and designing a flexible and scalable dataset design in-line with the application can optimize decision support process.

Many NoSQL systems enable client interaction through Representational State Transfer (REST) API. Protocol buffers (Protobuf) is a method for efficiently serializing and transmitting structured data. Serialization is to store not just the data but information about the data in stream of byte to help access to user-specific information across applications. JavaScript Object Notation (JSON) is a lightweight, text-based, open standard format for exchanging data between a server and a Web application. BSON is a format for binary-coded serialization of JSON-like documents (Gudivada et al., 2014) used in NoSQL.

NoSQL database environments are built with a distributed architecture so there are no single points of failure and there is built-in redundancy of both function and data. If one or more database servers, or 'nodes' goes down, the other nodes in the system are able to continue with operations without data loss, thereby showing true fault tolerance, location independence and continuous availability. The various types of NoSQL databases are:

- **Key-Value Store -** This kind of databases have a set of key- value pairs where key is unique and identifies a value. The key can be synthetic or auto-generated while the value can be String, JSON, BLOB (basic large object) etc. Limitations of this model is that it will not provide any kind of traditional database capabilities such as atomicity or consistency. Such capabilities must be provided by the application itself. As the volume of data increases, maintaining unique values as keys may become more difficult. Example: Riak and Amazon's Dynamo
- **Column-Family Database / Wide-Column Stores –** similar to key-value stores, except the keys point to multiple columns instead of rows, arranged by column family. Example: Cassandra, HBase
- **Document Store –**Pair each key with a complex data structure (usually in JSON or XML format) known as a document. These are essentially nested key-value stores, with keys being associated with nested values. Documents may

contain key-value pairs, key-array pairs or even nested documents. Example: MongoDB, CouchDB

- **Graph Database**-In some applications, relationships between objects are even more important than the objects themselves. Relationships can be static or dynamic. Such data is called connected data. Twitter, Facebook, Google, and LinkedIn data are naturally modeled using graphs.

The downside of using NoSQL databases are attributed to the fact that these databases lack proper standardized interfaces, they have low maturity compared to the relational databases and there is no proven and proper troubleshooting mechanisms and support from vendors.

Structuring unstructured big data in prominent nosql databases

NoSQL databases and management systems are relation- less (or schema-less). NoSQL databases are not "One size fits all" (i.e.) they are not based on a single model. They are designed to efficiently store significant amounts of unstructured data. This section explores how unstructured data are stored and retrieved in various NoSQL technologies like HBase (Column-Oriented database), MongoDB(Document database), OrientDB (Multi model database), ArangoDB (Multi model database).

A. HBase: It is a column-oriented database. It can manage structured and semi-structured data and has built-in features such as scalability, versioning, compression and garbage collection. Since it uses write-ahead logging and distributed configuration, it can provide fault-tolerance and quick recovery from individual server failures. HBase built on top of Hadoop / HDFS and the data stored in HBase can be manipulated using Hadoop's MapReduce capabilities.

Data in HBase is stored in Tables and these Tables are stored in Regions. When a Table becomes too big, the Table is partitioned into multiple Regions. These Regions are assigned to Region Servers across the cluster. Each Region Server contains a Write-Ahead Log (called HLog) and multiple Regions. Each Region in turn is made up of a MemStore and multiple StoreFiles (HFile). Where the data lives in the form of Column Families. The MemStore holds in-memory modifications to the Store data.

The mapping of Regions to Region Server is kept in a system table called .META. When trying to read or write data from HBase, the clients read the required Region information from the .META table and directly communicate with the appropriate Region Server. Each Region is identified by the start key (inclusive) and the end key (exclusive) .

HBase actually defines a four-dimensional data model and the following four coordinates define each cell.

- **Row Key:** Each row has a unique row key; the row key does not have a data type and is treated internally as a byte array.
- **Column Family:** Data inside a row is organized into column families; each row has the same set of column families, but across rows, the same column families do not need the same column qualifiers. Under-the-hood, HBase stores column families in their own data files, so they need to be defined upfront, and changes to column families are difficult to make.
- **Column Qualifier:** Column families define actual columns, which are called column qualifiers.
- **Version:** Each column can have a configurable number of versions, and you can access the data for a specific version of a column qualifier.

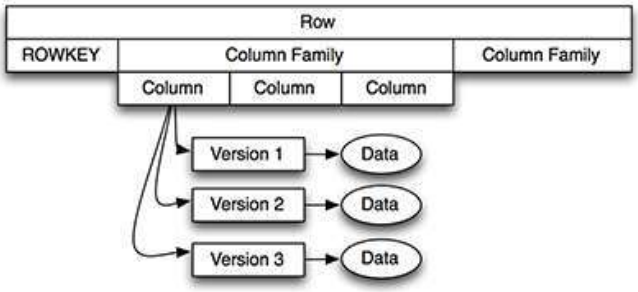

Fig 2. HBase Four-Dimensional Data Model

Source: http://www.informit.com/articles/article.aspx?p=2253412

HBase data can be accessed in two ways:

- Through their row key or via a table scan for a range of row keys
- In a batch manner using map-reduce

HBase addresses both a key/value store for real-time analysis and supports map-reduce for batch analysis.

B. MongoDB: This is made up of databases which contain collections of documents having fields. Collections can be indexed to improve lookup performance. The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. There are two tools that allow applications to represent these relationships: references and embedded documents.

References store the relationships between data by including links or references from one document to another. Applications can resolve these references to access the related data. Broadly, these are normalized data models.

Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures as sub-documents in a field or array within a document. These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.

If the document size exceeds the allocated space for that document, MongoDB will relocate the document on disk and adaptively adjusts the amount of automatic padding to reduce

occurrences of relocation. BSON is a binary serialization format used to store documents and make remote procedure calls in MongoDB. GridFS is a specification for storing and retrieving files that exceed the BSON-document size limit of 16MB.

Instead of storing a file in a single document, GridFS divides a file into parts, or chunks and stores each of those chunks as a separate document. By default GridFS limits chunk size to 255k. GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. While querying a GridFS for a file, the driver or client will reassemble the chunks as needed. It allows to perform range queries on files stored through GridFS and also access information from arbitrary sections of files, which allows you to "skip" into the middle of a video or audio file. GridFS is useful not only for storing files that exceed 16MB but also for storing any files for which you want access without having to load the entire file into memory.

C. OrientDB is a document-graph database (multi model database), Even if it is a document-based database, the relationships are managed as direct connections between records. OrientDB breaks the data into OO-like classes. Classes can be schema-less, schema-full or mixed. These classes have inheritance and in the graph world, can be either a (V)ertex or an (E)dge subclass. Each class has one or more clusters, which are "a generic way to group records". Classes can be grouped into clusters based on attribution.

For example, if there is Invoice class, you could group the 2014 invoices into a cluster Invoice2014 and the 2015 invoices into Invoice2015 cluster. You specify which cluster to use for a given record when you create that record. Every class has at least one default physical cluster that is used if none is specified on record creation. Records are an instance of a class. They are documents in the Document DB sense, as well as nodes in the Graph DB sense. Records live in a cluster and have the schema defined by the class. Much of a graph database's speed comes from the ease of traversing the graph from vertex to edge to vertex, etc. which is why graphs are the choice of most social networking sites. These kinds of relationships are meant to avoid joins.

OrientDB doesn't use JOINs. Instead it uses LINKs. A LINK is a relationship managed by storing the target RID in the source record. It's much like storing a pointer between two objects in memory. OrientDB uses a new indexing algorithm for both fast insertions and fast lookups.

- SB-Tree index, a new generation of algorithm designed to manage high number of concurrent clients. Furthermore it is durable by way of WAL (Write Ahead Logging) avoiding the need to rebuild indexes in case of failure.
- Hash Index, Based on hashing are super fast on read and write operations, but don't support range queries
- MVRB-Tree index, (Multi-Value-Red-Black Tree) as the best of both worlds between B+Tree and Red-Black Tree: fast insertion because it keeps the tree balanced with a maximum of 3 rotations in the worst case (similar to Red-Black Tree)

OrientDB's distributed architecture offers the flexibility of Multi-Master replication, where the database is replicated across a group of clustered computers (nodes) which all have access to updating the data. This can all be achieved concurrently by nodes modifying the data at the same time. OrientDB uses a Write Ahead Logging (WAL) Journal to make all the changes durable even in the event of failure.

D. ArangoDB: Multi-Model Database using flexible combinations of key-value pairs, documents and graphs.It basically provides access to documents which are stored in collections and are often called schemaless.

Documents in ArangoDB are uniquely identified by keys, allowing key/value storage. They can be connected, allowing to query them as graphs and even supports transactions with as many documents. A specialized binary data file format is used for disk storage. Structure data is stored separately from the documents and shared between documents with the same structure (i.e., with the same attribute names and types). Sharing the structure data between multiple documents greatly reduces disk storage space and memory usage for documents and is shared over three servers by a user defined or automatically chosen key.

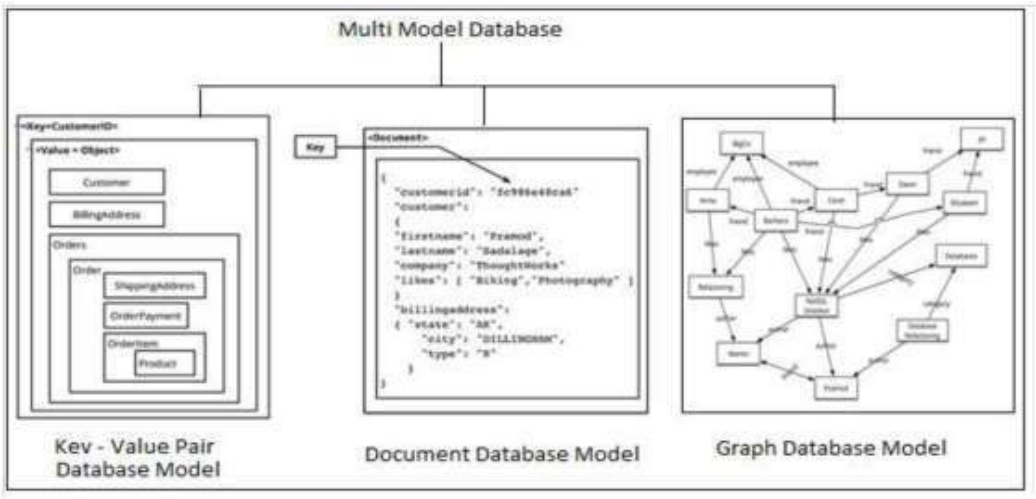

Fig 3. ArangoDB - Multi Model Database

By default the cluster uses the internal _key attribute as a sharding key. Alternatively you can choose one or more attributes of the collection as user defined sharding key. In case you want to use a unique index as a secondary index that attribute must be used as a sharding key.

It supports various indexes like "Full-text", "Geo", "Hash", "Bitmap" and "Skip-list". ArangoDB uses AppendOnly/MVCC to update documents by which locks are minimized and documents are quickly appended to the data files without having to reorganize or find empty spots, etc., and access to the DB is through the REST interface and there are no binary drivers.

ArangoDB can respond arbitrary HTTP requests directly through the Foxx application framework to aggregate data from multiple queries, to implement your own graph traversal algorithm millions of node and to modify many documents in a multi-collection transaction. This makes ArangoDB a combined database/application server.

AQL (ArangoDB Query Language) is a declarative query language and supports joins, graph queries, list iteration, results filtering, results projection, sorting, variables, grouping, aggregate functions, unions, and intersections. To access graph data ArangoDB uses "Path traversals"- small programs written in JavaScript. ArangoDB support the following ACID properties.

Conclusion

Given the continuing trend of data growth, new generation of solutions are required to cater to them. Data analytics over humongous data sets seems possible only by using distributed computation framework.A new generation of information management systems, termed NoSQL systems, caters to this trend and is apt for businesses that are planning to migrate existing applications to adapt to new trends of data growth, develop new applications involving unstructured data. Structuring such unstructured large heterogeneous in a standard mechanism will increase their usage and give greater insights hidden in them.

REFERENCES

Bednar, P., Sarnovsky, M., "RDF vs. NoSQL, databases for the semantic web applications Applied Machine Intelligence and Informatics (SAMI)," IEEE 12th International Symposium, Publication Year: 2014 , Page(s): 361 – 364.

Bernstein, "D. Today's Tidbit: VoltDB," Cloud Computing, IEEE Volume: 1, Issue: 1 Publication Year: 2014, Page(s): 90 – 92.

Brewer, E. A. July 2000. Towards Robust Distributed Systems. ACM Symposium on the Principles of Distributed Computing. Retrieved from http://www.cs. berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.

Gudivada, V.N., Rao, D., Raghavan, V.V. "NoSQL Systems for Big Data," Management in Proceedings of IEEE World Congress , Publication Year: 2014 , Page(s): 190 – 197.

http://docs.mongodb.org/master/MongoDB-data-models-guide.pdf

http://www.orientechnologies.com/docs/last/OrientDB-Manual.pdf

http://www.thoughtworks.com/insights/blog/nosql-databases-overview

https://docs.arangodb.com/manual.pdf

https://www.arangodb.com/2012/03/07/avocadodbs-design-objectives

Leavitt, N. "Will NoSQL Databases Live Up to Their Promise?," IEEE Volume: 43, Issue: 2 Publication Year: 2010 , Page(s): 12 – 14.

Man Qi, " Digital forensics and NoSQL databases", Fuzzy Systems and Knowledge Discovery (FSKD), 11th IEEE International Conference Publication Year: 2014, Page(s): 734 -739

XiaomingGao; Qiu, J. "Supporting Queries and Analyses of Large-Scale Social Media Data with Customizable and Scalable Indexing Techniques over NoSQL databases," Cluster, Cloud and Grid Computing (CCGrid), 14th IEEE/ACM International Symposium, Publication Year: 2014 , Page(s): 587 – 590.

****