© 2016 IJSRSET | Volume 2 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

NoSQL : A New Horizon in Big Data

Amarbir Singh
Department of Computer Science, Guru Nanak Dev University, Amritsar, Punjab, India

ABSTRACT

This Big data is useful for data sets where their size or type is away from the capability of traditional relational databases for capturing, managing and processing the data with low-latency. Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today. NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications. In this paper collection of NoSQL database tools are illustrated and also compared with the salient features.

Keywords: Big data, NoSQL, MangoDb, Data analysis, Data visualization, Redis

I. INTRODUCTION

Analyzing data can provide significant competitive advantage for an enterprise. The data when analyzed properly leads to a wealth of information which helps the businesses to redefine strategies. However the current volume of big data sets are too complicated to be managed and processed by conventional relational databases & data warehousing technologies. Using conventional techniques for Big Data storage and analysis is less efficient as memory access is slower. The data collection is also challenging as the volume and variety of data has to be derived from sources of different types [1].

This widespread demand for solutions, and the comparative ease of developing new systems, has led to a flowering of new databases. The main thing they have in common is that none of them support the traditional SQL interface, which has led to the movement being dubbed NoSQL. It's a bit misleading, though, since almost every production environment that they're used in also has an SQL-based database for anything that requires flexible queries and reliable transactions, and as the products mature, it's likely that some of them will start supporting the language as an option. If "NoSQL" seems too combative, think of it as "NotOnlySQL." These are all tools designed to trade the reliability and ease-of-use of traditional databases for the flexibility and performance required by new problems developers are encountering.

II. METHODS AND MATERIAL

1. Big Data Processing

Big data processing is achieved through kafka queues. Multiple data from various sources goes through the queue and is moved to either a NoSQL data store or HDFS. Depending on the data store we can write NoSQL queries or map reduce programs to extract the data and create reports for enabling business decisions as shown in Fig. 1. Performance measurement is important to support decision making and action taking in organizations [2]. It should be as dynamic as possible to keep pace with changes that happen in organizations. Therefore, it must be aligned to the organizations strategies and should be reviewed periodically [3].

Big data is not only large amount of data but there are various other features which create big difference between large amount of data and massive data. There are various definitions regarding big data and about how big data is viewed, all these definitions are discussed below categorically.

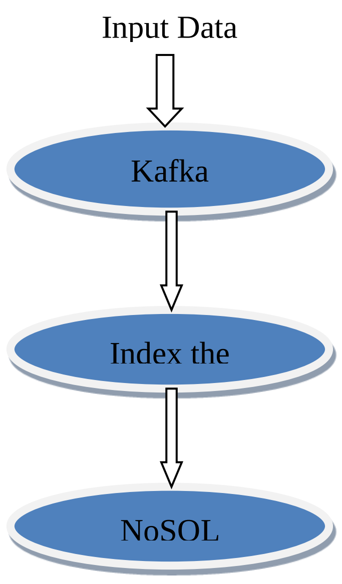

Figure 1: Big Data Processing

A. Attributive

As report given by IDC in 2011, big data extract the value from large amount of data by capturing high velocity data which further led to change of definition in big data which includes velocity, volume, variety and value. Further META group analyst noted big data as only three dimensional, velocity, volume and variety

B. Comparative

Mckinsey's report in 2011 defined bigdata as large data which makes it difficult to capture, store and analyze the data. This definition did not define bigdata properly. However it gave evolutionary aspect regarding bigdata

C. Architectural

As suggested by national institute of standard and technology (NIST), large velocity, volume and variety of data coined as bigdata limits its ability to use relational database system due to such features and require new technological means and horizontal scaling for processing and storing data [4]. Bigdata is further divided into two views as bigdata science and bigdata framework. Bigdata science covers techniques for evaluation of bigdata and bigdata framework covers algorithms and software libraries that help in distributed processing of bigdata across various clusters.

2. NOSQL Database Types

A. Document Databases

These databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

B. Graph Stores

These are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.

C. Key-value Stores

These databases are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or "key"), together with its value. Examples of key-value stores are Riak and Berkeley DB [5]. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.

D. Wide-column stores

This type databases such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

3. Various NoSQL Databases Tools

With so many different systems appearing, such a variety of design tradeoffs, and such a short track record for most, this list is inevitably incomplete and somewhat subjective. I'll be providing a summary of my own experiences with and impressions of each database, but I encourage you to check out their official web pages to get the most up-todate and complete view.

A. MongoDB

Mongo, whose name comes from "humongous", is a database aimed at developers with fairly large data sets, but who want something that's low maintenance and easy to work with. It's a document-oriented system, with records that look similar to JSON objects with the ability to store and query on nested attributes. From my own experience, a big advantage is the proactive support from the developers employed by 10gen, the commercial company that originated and supports the open source project [6]. I've always had quick and helpful responses both on the IRC channel and mailing list, something that's crucial when you're dealing with comparatively young technologies like these. It supports automatic sharding and MapReduce operations. Queries are written in JavaScript, with an interactive shell available, and bindings for all of the other popular languages.

B. CouchDB

CouchDB is similar in many ways to MongoDB, as a document-oriented database with a JavaScript interface, but it differs in how it supports querying, scaling, and versioning. It uses a multiversion concurrency control approach, which helps with problems that require access to the state of data at various times, but it does involve more work on the client side to handle clashes on writes, and periodic garbage collection cycles have to be run to remove old data. It doesn't have a good built-in method for horizontal scalability, but there are various external solutions like BigCouch, Lounge, and Pillow to handle splitting data and processing across a cluster of machines [7]. You query the data by writing JavaScript MapReduce functions called views, an approach that makes it easy for the system to do the processing in a distributed way. Views
offer a lot of power and flexibility, but they can be a bit overwhelming for simple queries.

C. Cassandra

Originally an internal Facebook project, Cassandra was open sourced a few years ago and has become the standard distributed database for situations where it's worth investing the time to learn a complex system in return for a lot of power and flexibility. Traditionally, it was a long struggle just to set up a working cluster, but as the project matures, that has become a lot easier.
It's a distributed key/value system, with highly structured values that are held in a hierarchy similar to the classic database/table levels, with the equivalents being keyspaces and column families. It's very close to the data model used by Google's BigTable, which you can find described in "BigTable" on page 8. By default, the data is sharded and balanced automatically using consistent hashing on key ranges, though other schemes can be configured. The data structures are optimized for consistent write performance, at the cost of occasionally slow read operations. One very useful feature is the ability to specify how many nodes must agree before a read or write operation completes. Setting the consistency level allows you to tune the CAP tradeoffs for your particular application, to prioritize speed over consistency or vice versa
The lowest-level interface to Cassandra is through Thrift, but there are friendlier clients available for most major languages. The recommended option for running queries is through Hadoop. You can install Hadoop directly on the same cluster to ensure locality of access, and there's also a distribution of Hadoop integrated with Cassandra available from DataStax [8]. There is a command-line interface that lets you perform basic administration tasks, but it's quite bare bones. It is recommended that you choose initial tokens when you first set up your cluster, but otherwise the decentralized architecture is fairly low-maintenance, barring major problems.

D. Redis

Two features make Redis stand out: it keeps the entire database in RAM, and its values can be complex data structures. Though the entire dataset is kept in memory, it's also backed up on disk periodically, so you can use it as a persistent database. This approach does offer fast and predictable performance, but speed falls off a cliff if the size of your data expands beyond available memory and the operating system starts paging virtual memory to handle accesses. This won't be a problem if you have small or predictably sized storage needs, but it does require a bit of forward planning as you're developing applications. You can deal with larger data sets by clustering multiple machines together, but the sharding is currently handled at the client level. There is an experimental branch of the code under active development that supports clustering at the server level. The support for complex data structures is impressive, with a large number of list and set operations handled quickly on the server side. It makes it easy to do things like appending to the end of a value that's a list, and then trim the list so that it only holds the most recent 100 items. These capabilities do make it easier to limit the growth of your data than it would be in most systems, as well as making life easier for application developers.

E. BigTable

BigTable is only available to developers outside Google as the foundation of the App Engine datastore. Despite

that, as one of the pioneering alternative databases, it's worth looking at.
It has a more complex structure and interface than many NoSQL datastores, with a hierarchy and multidimensional access. The first level, much like traditional relational databases, is a table holding data. Each table is split into multiple rows, with each row addressed with a unique key string. The values inside the row are arranged into cells, with each cell identified by a column family identifier, a column name, and a timestamp, each of which I'll explain below. The row keys are stored in ascending order within file chunks called shards. This ensures that operations accessing continuous ranges of keys are efficient, though it does mean you have to think about the likely order you'll be reading your keys in. Column names are confusingly not much like column names in a relational database [9]. They are defined dynamically, rather than specified ahead of time, and they often hold actual data themselves. If a column family represented inbound links to a page, the column name might be the URL of the page that the link is from, with the cell contents holding the link's text. The timestamp allows a given cell to have multiple versions over time, as well as making it possible to expire or garbage collect old data. A given piece of data can be uniquely addressed by looking in a table for the full identifier that conceptually looks like row key, then column family, then column name, and finally timestamp. You can easily read all the values for a given row key in a particular column family, so you could actually think of the column family as being the closest comparison to a column in a relational database. As you might expect from Google, BigTable is designed to handle very large data loads by running on big clusters of commodity hardware. It has per-row transaction guarantees, but it doesn't offer any way to atomically alter larger numbers of rows. It uses the Google File System as its underlying storage, which keeps redundant copies of all the persistent files so that failures can be recovered from.

F. HBase

HBase was designed as an open source clone of Google's BigTable, so unsurprisingly it has a very similar interface, and it relies on a clone of the Google File System called HDFS. It supports the same data structure of tables, row keys, column families, column names, timestamps, and cell values, though it is recommended that each table have no more than two or three families for performance reasons.

HBase is well integrated with the main Hadoop project, so it's easy to write and read to the database from a MapReduce job running on the system. One thing to watch out for is that the latency on individual reads and writes can be comparatively slow, since it's a distributed system and the operations will involve some network traffic. HBase is at its best when it's accessed in a distributed fashion by many clients. If you're doing serialized reads and writes you may need to think about a caching strategy.

G. Hypertable

Hypertable is another open source clone of BigTable. It's written in C++, rather than Java like HBase, and has focused its energies on high performance [10]. Otherwise, its interface follows in BigTable's footsteps, with the same column family and timestamping concepts.

H. Voldemort

An open source clone of Amazon's Dynamo database created by LinkedIn, Voldemort has a classic three-operation key/value interface, but with sophisticated backend architecture to handle running on large distributed clusters. It uses consistent hashing to allow fast lookups of the storage locations for particular keys, and it has versioning control to handle inconsistent values. A read operation may actually return multiple values for a given key if they were written by different clients at nearly the same time. This then puts the burden on the application to take some sensible recovery actions when it gets multiple values, based on its knowledge of the meaning of the data being written [11]. The example that Amazon uses is a shopping cart, where the set of items could be unioned together, losing any deliberate deletions but retaining any added items, which obviously makes sense—from a revenue perspective, at least!

I. Riak

Like Voldemort, Riak was inspired by Amazon's Dynamo database, and it offers a key/value interface and is designed to run on large distributed clusters. It also uses consistent hashing and a gossip protocol to avoid the need for the kind of centralized index server that

BigTable requires, along with versioning to handle update conflicts. Querying is handled using MapReduce functions written in either Erlang or JavaScript. It's open source under an Apache license, but there's also a closed source commercial version with some special features designed for enterprise customers.

J. ZooKeeper

When you're running a service distributed across a large cluster of machines, even tasks like reading configuration information, which are simple on single-machine systems, can be hard to implement reliably. The ZooKeeper framework was originally built at Yahoo! to make it easy for the company's applications to access configuration information in a robust and easy-to-understand way, but it has since grown to offer a lot of features that help coordinate work across distributed clusters. One way to think of it is as a very specialized key/value store, with an interface that looks a lot like a filesystem and supports operations like watching callbacks, write consensus, and transaction IDs that are often needed for coordinating distributed algorithms.

This has allowed it to act as a foundation layer for services like LinkedIn's Norbert, a flexible framework for managing clusters of machines. ZooKeeper itself is built to run in a distributed way across a number of machines, and it's designed to offer very fast reads, at the expense of writes that get slower the more servers are used to host the service [12].

III. RESULTS AND DISCUSSION

Comparison of SQL and NoSQL Databases

Various general and development features of SQL and NoSQL databases are compared in table 1. This comparison highlights the importance of NOSQL databases in current big data scenario.

Table 1 Comparison of SQL and NoSQL Databases

Features	SQL Databases	NOSQL Databases
Types of Databases	One type (SQL database) with minor variations	Many different types including key-value stores, document databases, wide-column stores, and graph databases
Development History	Developed in 1970s to deal with first wave of data storage applications	Developed in late 2000s to deal with limitations of SQL databases, especially scalability, multi-structured data, geo-distribution and agile development sprints
Examples	MySQL, Postgres, Microsoft SQL Server, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Data Storage Model	Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Related data is stored in separate tables, and then joined together when more complex queries are executed. For example, "offices" might be stored in one table, and "employees" in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables together to get all the information necessary.	Varies based on database type. For example, key-value stores function similarly to SQL databases, but has only two columns ("key" and "value"), with more complex information sometimes stored as BLOBs within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic, with some enforcing data validation rules. Applications can add new fields on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For

		some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required, and core relational features such as JOINs, referential integrity and transactions are typically lost.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary.
Development Model	Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open-source
Supports Transactions	Yes, updates can be configured to complete entirely or not at all	In certain circumstances and at certain levels (e.g., document level vs. database level)
Data Manipulation	Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE…	Through object-oriented APIs
Consistency	Can be configured for strong consistency	Depends on product. Some provide strong consistency (e.g., MongoDB, with tunable consistency for reads) whereas others offer eventual consistency (e.g., Cassandra).

IV. CONCLUSION

This paper provides an introduction to big data and various database tools that are currently being used by the developers along with the comparison of SQL and NoSQL database tools. NoSQL databases are rapidly replacing the SQL databases in big data scenario due to the no of advantages they provide. A large number of NoSQL database tools have emerged in a short span of time and it seems that this trend will continue in the time to come.

V. REFERENCES

[1] Sagiroglu, S.; Sinanc, D. ,(20-24 May 2013),"Big Data: A Review"

[2] Chris Deptula, "With all of the Big Data Tools, what is the right one for me", www.openbi.com/blogs/chris%20Deptula,retrieved 08/02/14.

[3] http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data.

[4] Garlasu, D.; Sandulescu, V. ; Halcu, I. ; Neculoiu, G. ;,(17-19 Jan. 2013),"A Big Data implementation based on Grid Computing", Grid Computing

[5] Mukherjee, A.; Datta, J.; Jorapur, R.; Singhvi, R.; Haloi, S.; Akram, W., (18-22 Dec.,2012) , "Shared disk big data analytics with Apache Hadoop"

[6] http://dashburst.com/infographic/big-data-volume.var iety-velocity/

[7] http://www01.ibm.com/software/in/data/bigdata/ Mark Troester(2013), "Big Data Meets Big Data Analytics", www.sas.com/resources/.../WR46345.pdf, retrieved 10/02/14.

[8] Brian Runciman(2013), "IT NOW Big Data Focus, AUTUMN 2013", www.bcs.org, retrieved 03/02/14.

[9] Neil Raden (2012), " Big Data Analytics and complete Architecture",www.teradata.com/ Big-Data Analytics", retrieved 15/03/14.

[10] K. Bakshi, "Considerations for Big Data: Architecture and Approach", Aerospace Conference IEEE, Big Sky Montana, March 2012.

[11] Sagiroglu, S.; Sinanc, D. ,(20-24 May 2013),"Big Data: A Review"

[12] Grosso, P. ; de Laat, C. ; Membrey, P.,(20-24 May 2013)," Addressing big data issues in Scientific Data Infrastructure"