ISSN:0975-9646

Big Data Analytics: Hadoop-Map Reduce & NoSQL Databases

Abinav Pothuganti

Computer Science and Engineering, CBIT,Hyderabad, Telangana, India

***Abstract*— Today, we are surrounded by data like oxygen. The exponential growth of data first presented challenges to cutting- edge businesses such as Google, Yahoo, Amazon, Microsoft, Facebook, Twitter etc. Data volumes to be processed by cloud applications are growing much faster than computing power. This growth demands new strategies for processing and analysing information. Such large volume of un-structured (or semi structured) and structured data that gets created from various applications, emails, web logs, social media is known as "Big Data". This kind of data exceeds the processing capacity of conventional database systems. In this paper we will provide the basic knowledge about Big Data, which is majorly being generated because of cloud computing and also explain in detail about the two widely used Big Data Analytics techniques i.e. Hadoop MapReduce and NoSQL Database.**

***Keywords*— Big Data, Big Data Analytics, Hadoop, NoSQL Introduction**

I. INTRODUCTION

Cloud computing has been driven fundamentally by the need to process an exploding quantity of data in terms of exabytes as we are approaching the Zetta Byte Era. One critical trend shines through the cloud is Big Data. Indeed, it's the core driver in cloud computing and will define the future of IT. When a company needed to store and access more data they had one of two choices. One option would be to buy a bigger machine with more CPU, RAM, disk space, etc. This is known as scaling vertically. Of course, there is a limit to how big of a machine you can actually buy and this does not work when you start talking about internet scale. The other option would be to scale horizontally. This usually meant contacting some database vendor to buy a bigger solution. These solutions do not come cheap and therefore required a significant investment. Today, the source of data generated not only by the users and applications but also "machine- generated," and such data is exponentially leading the change in the Big Data space.

Big Data processing is performed through a programming paradigm known as MapReduce. Typically, implementation of the MapReduce paradigm requires networked attached storage and parallel processing. The computing needs of MapReduce programming are often beyond what small and medium sized business are able to commit.

Cloud computing is on-demand network access to Computing resources, provided by an outside entity. Common deployment models for cloud computing include platform as a service (PaaS), software as a service (SaaS), infrastructure as a service (IaaS), and hardware as a service (HaaS).

Platform as a Service (PaaS) is the use of cloud computing to provide platforms for the development and use of custom applications. Software as a service (SaaS) provides businesses with applications that are stored and run on virtual servers – in the cloud. In the IaaS model, a client business will pay on a per-use basis for use of equipment to support computing operations including storage, hardware, servers, and networking equipment. HaaS is a cloud service based upon the model of time sharing on minicomputers and mainframes.

The three types of cloud computing are the public cloud, the private cloud, and the hybrid cloud. A public cloud is the pay- as-you-go services. A private cloud is internal data center of a business not available to the general public but based on cloud structure. The hybrid cloud is a combination of the public cloud and private cloud.

Three major reasons for small to medium sized businesses to use cloud computing for big data technology implementation are hardware cost reduction, processing cost reduction, and ability to test the value of big data.

II. BIG DATA

Big data is a collection of data sets so large and complex which is also exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the structures of our current database architectures. Big Data is typically large volume of un-structured (or semi structured) and structured data that gets created from various organized and unorganized applications, activities and channels such as emails, tweeter, web logs, Facebook, etc. The main difficulties with Big Data include capture, storage, search, sharing, analysis, and visualization. The core of Big Data is Hadoop which is a platform for distributing computing problems across a number of servers. It is first developed and released as open source by Yahoo!, it implements the MapReduce approach pioneered by Google in compiling its search indexes. Hadoop's MapReduce involves distributing a dataset among multiple servers and operating on the data: the "map" stage. The partial results are then recombined: the "reduce" stage. To store data, Hadoop utilizes its own distributed file system, HDFS, which makes data available to multiple computing nodes. Big data explosion, a result not only of increasing Internet usage by people around the world, but also the connection of billions of devices to the Internet. Eight years ago, for example, there were only around 5 exabytes of data online. Just two years ago, that

amount of data passed over the Internet over the course of a single month. And recent estimates put monthly Internet data flow at around 21 exabytes of data. This explosion of data - in both its size and form - causes a multitude of challenges for both people and machines.

III. HADOOP MAP REDUCE

Hadoop is a batch processing system for a cluster of nodes that provides the underpinnings of most Big Data analytic activities because it bundle two sets of functionality most needed to deal with large unstructured datasets namely, Distributed file system and MapReduce processing. It is a project from the Apache Software Foundation written in Java to support data intensive distributed applications. Hadoop enables applications to work with thousands of nodes and petabytes of data. The inspiration comes from Google's MapReduce and Google File System papers. Hadoop's biggest contributor has been the search giant Yahoo, where Hadoop is extensively used across the business platform.

A. High Level Architecture of Hadoop

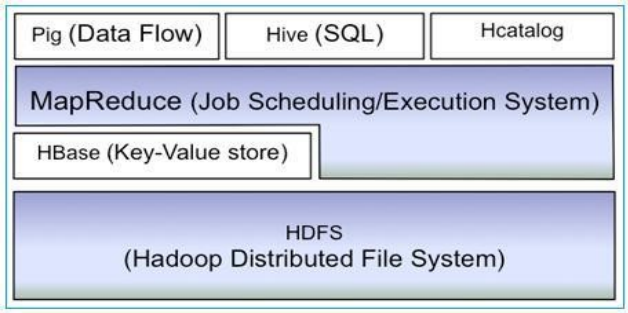

Fig 1. High level Architecture of Hadoop

Pig: It is a dataflow processing (scripting) language Apache Pig is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs. The main characteristic of Pig programs is that their structure can be substantially parallelized enabling them to handle very large data sets, simple syntax and advanced built-in functionality provide an abstraction that makes development of Hadoop jobs quicker and easier to write than traditional Java MapReduce jobs.

Hive: Hive is a data warehouse infrastructure built on top of Hadoop. Hive provides tools to enable easy data summarization, ad-hoc querying and analysis of large datasets stored in Hadoop files. It provides a mechanism to put structure on this data and it also provides a simple query language called Hive QL, based on SQL, enabling users familiar with SQL to query this data.

HCatalog: It is a storage management layer for Hadoop that enables users with different data processing tools. HCatalog's table abstraction presents users with a relational view of data in the Hadoop distributed file system (HDFS) and ensures that users need not worry about where or in what format their data is stored.

MapReduce: Hadoop MapReduce is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of computer nodes. MapReduce uses the HDFS to access file segments and to store reduced results.

HBase: HBase is a distributed, column-oriented database. HBase uses HDFS for its underlying storage. It maps HDFS data into a database like structure and provides Java API access to this DB. It supports batch style computations using MapReduce and point queries (random reads). HBase is used in Hadoop when random, realtime read/write access is needed. Its goal is the hosting of very large tables running on top of clusters of commodity hardware.

HDFS: Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS is, as its name implies, a distributed file system that provides high throughput access to application data creating multiple replicas of data blocks and distributing them on compute nodes throughout a cluster to enable reliable and rapid computations.

Core: The Hadoop core consist of a set of components and interfaces which provides access to the distributed file systems and general I/O (Serialization, Java RPC, Persistent data structures). The core components also provide "Rack Awareness", an optimization which takes into account the geographic clustering of servers, minimizing network traffic between servers in different geographic clusters.

B. Architecture of Hadoop

Hadoop is a Map/Reduce framework that works on HDFS or on HBase. The main idea is to decompose a job into several and identical tasks that can be executed closer to the data (on the DataNode). In addition, each task is parallelized: the Map phase. Then all these intermediate results are merged into one result: the Reduce phase. In Hadoop, The JobTracker (a java process) is responsible for monitoring the job, managing the Map/Reduce phase, managing the retries in case of errors. The TaskTrackers (Java process) are running on the different DataNodes. Each TaskTracker executes the tasks of the job on the locally stored data.

The core of the Hadoop Cluster Architecture is given below:

HDFS (Hadoop Distributed File System): HDFS is the basic file storage, capable of storing a large number of large files.

MapReduce: MapReduce is the programming model by which data is analyzed using the processing resources within the cluster.

Each node in a Hadoop cluster is either a master or a slave. Slave nodes are always both a Data Node and a Task Tracker. While it is possible for the same node to be both a Name Node and a JobTracker

Name Node: Manages file system metadata and access control. There is exactly one Name Node in each cluster.

Secondary Name Node: Downloads periodic checkpoints from the name Node for fault-tolerance. There is exactly one Secondary Name Node in each cluster.

Job Tracker: Hands out tasks to the slave nodes. There is exactly one Job Tracker in each cluster.

Data Node: Holds file system data. Each data node manages its own locally-attached storage and stores a copy

of some or all blocks in the file system. There are one or more Data Nodes in each cluster.

Task Tracker: Slaves that carry out map and reduce tasks. There are one or more Task Trackers in each cluster.

C. Hadoop Distributed File System (HDFS)

An HDFS cluster has two types of node operating in a master-worker pattern: a NameNode (the master) and a number of DataNodes (workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. The namenode also knows the datanodes on which all the blocks for a given file are located. Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing. Name Node decides about replication of data blocks. In a typical HDFS, block size is 64MB and replication factor is 3 (second copy on the local rack and third on the remote rack). The Figure 4 shown architecture distributed file system HDFS. Hadoop MapReduce applications use storage in a manner that is different from general-purpose computing. To read an HDFS file, client applications simply use a standard Java file input stream, as if the file was in the native filesystem. Behind the scenes, however, this stream is manipulated to retrieve data from HDFS instead. First, the Name Node is contacted to request access permission. If granted, the Name Node will translate the HDFS filename into a list of the HDFS block IDs comprising that file and a list of Data Nodes that store each block, and return the lists to the client. Next, the client opens a connection to the "closest" Data Node (based on Hadoop rack-awareness, but optimally the same node) and requests a specific block ID. That HDFS block is returned over the same connection, and the data delivered to the application. To write data to HDFS, client applications see the HDFS file as a standard output stream. Internally, however, stream data is first fragmented into HDFS-sized blocks (64MB) and then smaller packets (64kB) by the client thread. Each packet is enqueued into a FIFO that can hold up to 5MB of data, thus decoupling the application thread from storage system latency during normal operation. A second thread is responsible for dequeuing packets from the FIFO, coordinating with the Name Node to assign HDFS block IDs and destinations, and transmitting blocks to the Data Nodes (either local or remote) for storage. A third thread manages acknowledgements from the Data Nodes that data has been committed to disk.

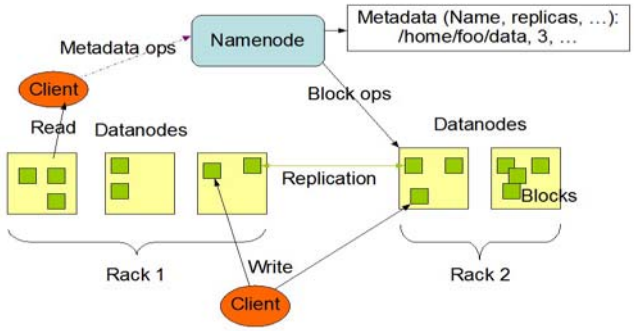

Fig 2. Hadoop Distributed Cluster File System Architecture

D. Map Reduce Architecture & Implementation

MapReduce is a data processing or parallel programming model introduced by Google. In this model, a user specifies the computation by two functions, Map and Reduce. In the mapping phase, MapReduce takes the input data and feeds each data element to the mapper. In the reducing phase, the reducer processes all the outputs from the mapper and arrives at a final result. In simple terms, the mapper is meant to filter and transform the input into something that the reducer can aggregate over. The underlying MapReduce library automatically parallelizes the computation, and handles complicated issues like data distribution, load balancing and fault tolerance. Massive input, spread across many machines, need to parallelize. Moves the data, and provides scheduling, fault tolerance. The original MapReduce implementation by Google, as well as its open-source counterpart, Hadoop, is aimed for parallelizing computing in large clusters of commodity machines. Map Reduce has gained a great popularity as it gracefully and automatically achieves fault tolerance. It automatically handles the gathering of results across the multiple nodes and returns a single result or set.

MapReduce model advantage is the easy scaling of data processing over multiple computing nodes.

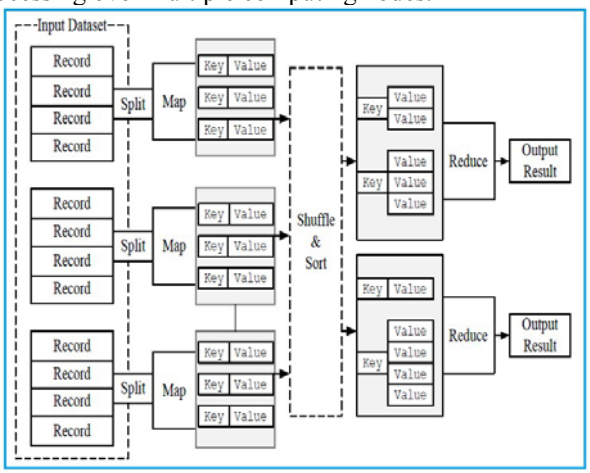

Fig 3. High Level view of MapReduce Programming Model

Fault tolerance: MapReduce is designed to be fault tolerant because failures are common phenomena in large scale distributed computing and it includes worker failure and master failure.

Worker failure: The master pings every mapper and reducer periodically. If no response is received for a certain amount of time, the machine is marked as failed. The ongoing task and any tasks completed by this mapper will be re-assigned to another mapper and executed from the very beginning. Completed reduce tasks do not need to be re-executed because their output is stored in the global file system.

Master failure: Since the master is a single machine, the probability of master failure is very small. MapReduce will re- start the entire job if the master fails. There are currently three popular implementations of the MapReduce programming model namely Google MapReduce, Apache Hadoop, Stanford Phoenix.

E. Execution Process in MapReduce Programming Model

In MapReduce programming model and a MapReduce job consists of a map function, a reduce function, and When a function called the below steps of actions take place:

- MapReduce will first divide the data into N partitions with size varies from 16MB to 64MB
- Then it will start many programs on a cluster of different machines. One of program is the master program; the others are workers, which can execute their work assigned by master. Master can distribute a map task or a reduce task to an idle worker.
- If a worker is assigned a Map task, it will parse the input data partition and output the key/value pairs, then pass the pair to a user defined Map function. The map function will buffer the temporary key/value pairs in memory. The pairs will periodically be written to local disk and partitioned into P pieces. After that, the local machine will inform the master of the location of these pairs.
- If a worker is assigned a Reduce task and is informed about the location of these pairs, the Reducer will read the entire buffer by using remote procedure calls. After that, it will sort the temporary data based on the key.
- Then, the reducer will deal with all of the records. For each key and according set of values, the reducer passes key/value pairs to a user defined Reduce function. The output is the final output of this partition.
- After all of the mappers and reducers have finished their work, the master will return the result to users' programs. The output is stored in F individual files.

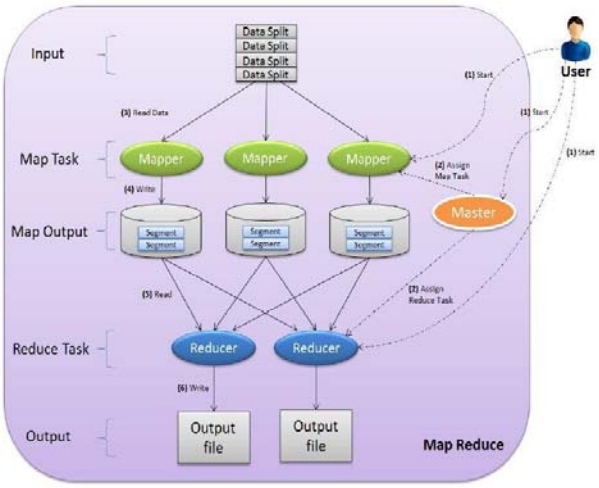

Fig 4. Architecture of MapReduce

F. A MapReduce Programming Model Example

In essence MapReduce is just a way to take a big task and split it into discrete task that can be done in parallel. A simple problem that is often used to explain how MapReduce works in practice consists in counting the occurrences of single words within a text. This kind of problem can be easily solved by launching a single MapReduce job as given in the below:

- Input data
- Input data are partitioned into smaller chunks of data
- For each chunk of input data, a "map task" runs which applies the map function resulting output of each map task is a collection of key-value pairs.
- The output of all map tasks is shuffled for each distinct key in the map output; a collection is created containing all corresponding values from the map output.
- For each key-collection resulting from the shuffle phase, a "reduce task" runs which applies the reduce function to the collection of values.
- The resulting output is a single key-value pair.
- The collection of all key-value pairs resulting from the reduce step is the output of the MapReduce job.

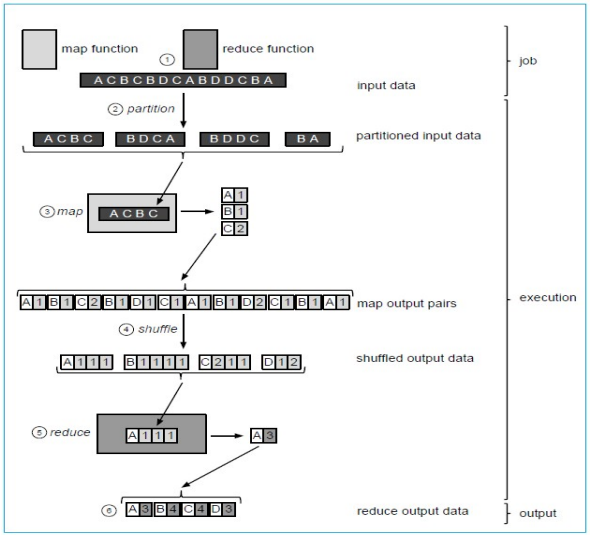

Fig 5. A MapReduce Programming Model Example

IV. NoSQL Databases

NoSQL systems are distributed, non-relational databases designed for large-scale data storage and for massively- parallel data processing across a large number of commodity servers. They also use non-SQL languages and mechanisms to interact with data (though some new feature APIs that convert SQL queries to the system's native query language or tool). NoSQL database systems arose alongside major Internet companies, such as Google, Amazon, and Facebook; which had challenges in dealing with huge quantities of data with conventional RDBMS solutions could not cope.

A. *Evolution of NoSQL Databases*

Of the many different data-models, the relational model has been dominating since the 80s, with implementations like Oracle databases, MySQL and Microsoft SQL Servers-also known as Relational Database Management System (RDBMS). Lately, however, in an increasing number of cases the use of

relational databases leads to problems both because of deficits and problems in the modelling of data and constraints of horizontal scalability over several servers and big amounts of data. There are two trends that bringing these problems to the attention of the international software community:

1. The exponential growth of the volume of data generated by users, systems and sensors, further accelerated by the concentration of large part of this volume on big distributed systems like Amazon, Google and other cloud services.
2. The increasing interdependency and complexity of data accelerated by the Internet, Web2.0, social networks and open and standardized access to data sources from a large number of different systems.

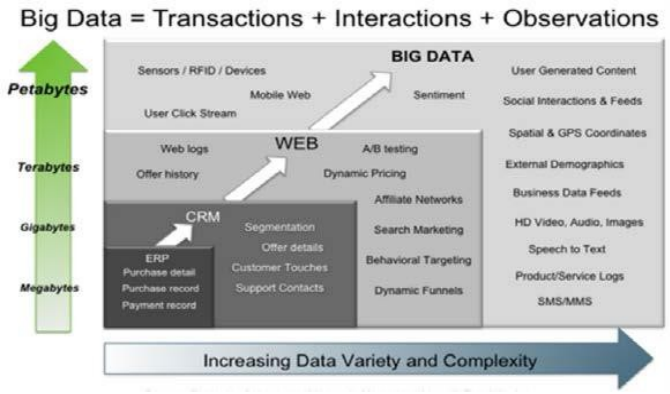

Fig 6. Big Data Transactions with Interactions and Observations

Organizations that collect large amounts of unstructured data are increasingly turning to non-relational databases, now frequently called NoSQL databases. NoSQL databases focus on analytical processing of large scale datasets, offering increased scalability over commodity hardware. Computational and storage requirements of applications such as for Big Data Analytics, Business Intelligence and social networking over peta-byte datasets have pushed SQL-like centralized databases to their limits. This led to the development of horizontally scalable, distributed non-relational data stores, called No-SQL databases.

B. *Characteristics of NoSQL Databases*

In order to guarantee the integrity of data, most of the classical database systems are based on transactions. This ensures consistency of data in all situations of data management. These transactional characteristics are also known as ACID (Atomicity, Consistency, Isolation, and Durability).However, scaling out of ACID-compliant systems has shown to be a problem. Conflicts are arising between the different aspects of high availability in distributed systems that are not fully solvable - known as the CAP- theorem.

Strong Consistency: all clients see the same version of the data, even on updates to the dataset - e. g., by means of the two-phase commit protocol (XA transactions), and ACID.

High Availability: all clients can always find at least one copy of the requested data, even if some of the machines in a cluster are down.

Partition-tolerance: the total system keeps its characteristic even when being deployed on different servers, transparent to the client.

The CAP-Theorem postulates that only two of the three different aspects of scaling out are can be achieved fully at the same time.

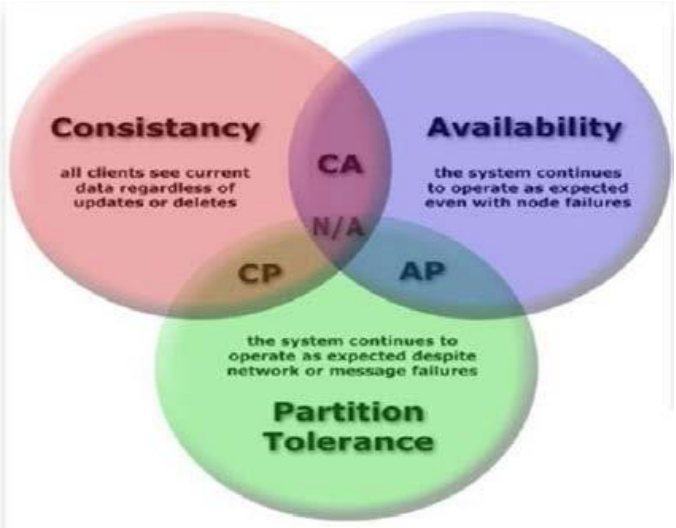

Fig 7. Characteristics of NoSQL Databases

C. *Classification of NoSQL Databases*

We classify NoSQL Databases in four basic categories, each suited to different kinds of tasks:

- Key-Value stores
- Document databases (or stores)
- Wide-Column (or Column-Family) stores
- Graph databases.

Key-Value stores:

Typically, these DMS store items as alpha-numeric identifiers (keys) and associated values in simple, standalone tables (referred to as "hash tables"). The values may be simple text strings or more complex lists and sets. Data searches can usually only be performed against keys, not values, and are limited to exact matches.

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto

Fig 8. Key/Value Store NoSQL Database

Document Databases:

Inspired by Lotus Notes, document databases were, as their name implies, designed to manage and store documents. These documents are encoded in a standard data

exchange format such as XML, JSON (Javascript Option Notation) or BSON (Binary JSON). Unlike the simple key-value stores described above, the value column in document databases contains semi- structured data-specifically attribute name/value pairs. A single column can house hundreds of such attributes, and the number and type of attributes recorded can vary from row to row.

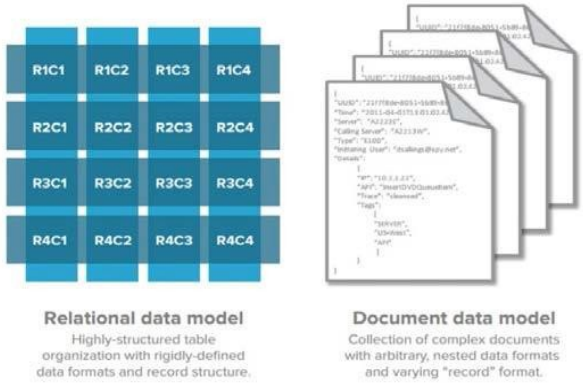

Fig 9. Document Store NoSQL Database

Wide-Column (or Column-Family) Stores:

Like document databases, Wide-Column (or Column-Family) stores (hereafter WC/CF) employ a distributed, column-oriented data structure that accommodates multiple attributes per key. While some WC/CF stores have a Key-Value DNA (e.g., the Dynamo-inspired Cassandra), most are patterned after Google's Bigtable, the petabyte-scale internal distributed data storage system Google developed for its search index and other collections like Google Earth and Google Finance. These generally replicate not just Google's Bigtable data storage structure, but Google's distributed file system (GFS) and MapReduce parallel processing framework as well, as is the case with Hadoop, which comprises the Hadoop File System (HDFS, based on GFS) + Hbase (a Bigtable style storage system) + MapReduce.

Graph Databases:

Graph databases replace relational tables with structured relational graphs of interconnected key-value pairings. They are similar to object-oriented databases as the graphs are represented as an object-oriented network of nodes (conceptual objects), node relationships ("edges") and properties (object attributes expressed as key-value pairs). They are the only of the four NoSQL types discussed here that concern themselves with relations, and their focus on visual representation of information makes them more human- friendly than other NoSQL DMS.

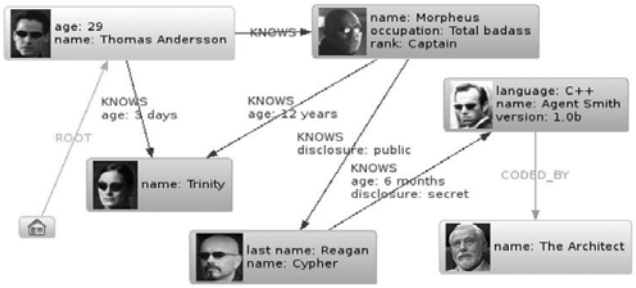

Fig 10. Graph NoSQL Database

V. Conclusion

Cloud technology progress & increased use of the Internet are creating very large new datasets with increasing value to businesses and processing power to analyse them affordable. Big Data is still in its early infancy but it is already having a profound effect on technology companies and the way we do business. The size of these datasets suggests that exploitation may well require a new category of data storage and analysis systems with different architectures.

Both Hadoop-MapReduce programming paradigm and NoSQL Databases have a substantial base in the Big Data community and they are still expanding. HDFS, the Hadoop Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high-throughput access to these information.

Ease-of-use of the MapReduce method has made the Hadoop Map Reduce technique more popular than any other Data Analytics techniques (methods).

References

[1] R. Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in *bioinformatics BMC bioinformatics,11(Suppl 12):*S1, 2010.

[2] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.

[3] HDFS (hadoop distributed file system) architecture. http://hadoop.apache.org/common/docs/current/hdfs

[4] R. Lammel. Google"s mapreduce programming model – revisited. Science of Computer Programming, 70(1):1 – 30, 2008.

[5] A Workload Model for MapReduce by Thomas A. de Ruiter, Master"s Thesis in *Computer Science, Parallel and Distributed Systems Group Faculty of Electrical Engineering, Mathematics, and Computer Science. Delft University of Technology*, 2nd June 2012.

[6] Dhruba Borthaku, *The Hadoop Distributed File System: Architecture and Design*. Retrieved from, 2010, http://hadoop.apache.org/common/.

[7] Y. Hung-Chih, D. Ali, H. Ruey-Lung, and D.S. Parker, *"Map-Reduce-Merg e: Simplified Relation al Data Processing On Large Clusters"*, in Proceedings of the 2007 ACM Sigmod International Conference on Management of Data Beijing", China: acm, 2007.

[8] Rabi Prasad Padhy, *"Big Data Processing with Hadoop-MapReduce in Cloud Systems"* in International Journal of Cloud Computing and Services Science

[9] A B M Moniruzzaman and Syed Akhter Hossain, *"NoSQL Database: New Era of Databases for Big data Analytics-Classification, Characteristics and Comparison"*, in International Journal of Database Theory and Application