Data Science Journal, Volume 13, 4 December 2014

A BRIEF REVIEW ON LEADING BIG DATA MODELS

Sugam Sharma[1*], Udoyara S Tim[2], Johnny Wong[3], Shashi Gadia[3], Subhash Sharma[4]

[1] Center for Survey Statistics and Methodology, Iowa State University, Ames, IA, 50010, USA
**Email:* sugamsha@iastate.edu
[2] Department of Agricultural and Biosystems Engineering, Iowa State University, Ames, IA, 50010, USA
[3] Department of Computer Science, Iowa State University, Ames, IA, 50010, USA
[4] Electronics & Computer Discipline, DPT, Indian Institute of Technology, Roorkee, 247001, INDIA

ABSTRACT

Today, science is passing through an era of transformation, where the inundation of data, dubbed data deluge is influencing the decision making process. The science is driven by the data and is being termed as data science. In this internet age, the volume of the data has grown up to petabytes, and this large, complex, structured or unstructured, and heterogeneous data in the form of "Big Data" has gained significant attention. The rapid pace of data growth through various disparate sources, especially social media such as Facebook, has seriously challenged the data analytic capabilities of traditional relational databases. The velocity of the expansion of the amount of data gives rise to a complete paradigm shift in how new age data is processed. Confidence in the data engineering of the existing data processing systems is gradually fading whereas the capabilities of the new techniques for capturing, storing, visualizing, and analyzing data are evolving. In this review paper, we discuss some of the modern Big Data models that are leading contributors in the NoSQL era and claim to address Big Data challenges in reliable and efficient ways. Also, we take the potential of Big Data into consideration and try to reshape the original operational-oriented definition of "Big Science" (Furner, 2003) into a new data-driven definition and rephrase it as "The science that deals with Big Data is Big Science."

Keywords: Big Data, Challenges, Models, NoSQL, Big Science

1 INTRODUCTION

In recent years, large spatial, temporal, and spatio-temporal databases have gained considerable attention and momentum. With the rapid growth in the Geographical Information System (GIS), the result of the amount of data collected by various agencies is the mammoth heterogeneous data available. Statistics (Demiryurek et al., 2010) reveal that about 80% of all stored data in corporate databases are spatial data. In the last two years, the rampant usage of social media such as Twitter, Facebook, etc. is the sole contributor of approximately 90% of the total data available today, and this huge amount of potentially important data is stored at various distributed locations and in heterogeneous formats. Though the arena of data is growing at a very fast pace in its breadth and depth, their management techniques are not advancing at the same pace, resulting in the scarcity of suitably efficient data processing systems. Most of the current database management systems originated in the 1970s, and 1980s though many of them have experienced significant remodeling (Carino & Sterling, 1998) including the query structure.

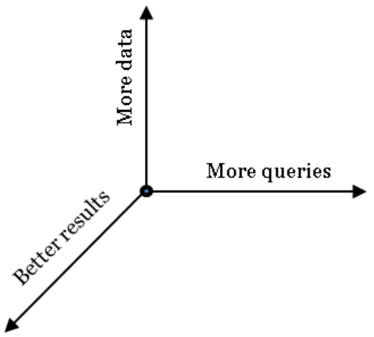


Figure 1. Data philosophy

Growing challenges due to severe data proliferation, increasing shallowness in confidence in the existing data models and their querying capacity, the sluggish emergence of capable data models, and the motivation for data-driven analytics are critical concerns that demand timely action, attention, and federated endeavors from scientific communities. In sum, the scientific, industrial, or institutional understanding of the term "data" has completely been reclassified into what is popularly being referred as “Big Data” today. Based upon the ubiquitous nature of Big Data, we redevelop the original operational-oriented definition of “Big Science” (Furner, 2003) into a new data-centric definition as “The science that deals with Big Data is Big Science.”

Even though over the past few years, a few robust Big Data models have come into existence, there is still a need for the pool to expand at a faster pace to meet the challenges of data proliferation. Apparently, every data service sees continuing growth in computational needs. Figure 1 is a three dimensional abstraction of the data philosophy at this very hour, and each dimension can be understood as following. Over the last few years, the number of internet users, especially in the social media, has grown enormously with the result of very face-paced, massive and complex data production. The scope of this production requires more efficient querying development to retrieve more accurate information (better results) within the shortest time frame. These challenges demand the expansion of the existing infrastructure into a large, efficient, and scalable infrastructure.

In this paper, we briefly review some newly conceived data models that are at the forefront of conversations concerning handling Big Data challenges. We discuss six data models and their querying systems that claim to be capable of handling Big Data of a size beyond a petabyte magnitude and are leading contributors to the NoSQL (largely being translated as “not only SQL”) age. Important requirements of the storage technology of Big Data models encompass a few primary characteristics such large size, scalable, availability, distributed, etc. The models are conceptualized to be robust, schema-free, non-relational, highly scalable, efficiently distributed, highly available, easily replicated, and largely (as of now) open-source. The paper is structured as follows. Section 2 discusses Big Data and its current challenges. Section 3 briefly reviews some of the leading modern NoSQL data models that are meeting these Big Data challenges. Section 4 summarizes the discussion and comparison of the models. The paper’s limitations are discussed in Section 5, and Section 6 contains concluding remarks.

2 BIG DATA AND CHALLENGES

The age of data science is in its infancy and is experiencing a tactical evolution by leaps and bound in all dimensions of science. The digital footprints and sizeable traces of the data are being produced and deposited in massive quantities by various data outlet harbingers such as Facebook, Twitter, Wikipedia, and Google to name a few (Villars et al., 2011; IBM, 2012). Scientists from almost every community express significant interest and eagerness in accessing these raw data to facilitate optimal decision making. Disparate scientific communities wish to use these data but argue about accessibility and are looking for new technologies to enhance access to the data. The fast-paced growth of modern data - structured and unstructured - raises doubt about the computing capacity of existing database models. In the last few years, a variety of sources, such as sensors gathering climatic information, power meter readings, social media sites, videos and digital pictures, and traffic data, have been responsible for the rapid growth of voluminous data - about 2.5 quintillion bytes every day (IBM, 2012). Because of the fast growth of the data its sheer volume is a central issue today. About 90% of the data existing in 2012 was produced in the previous two years (IBM, 2012). Existing database management systems fall short not only in storage capacity but also in data processing. In this era of data science, the term “data” is further redefined as “Big Data”. This may help to convincingly reformulate the definition of “Big Science” (Furner, 2003) into a data-centric definition - “The science that deals with Big Data is Big Science.” The concept of Big Data is relatively new and needs further research. This section highlights some important aspects about Big Data. Primarily, Big Data is described as having five concerns: data volume, velocity, variety, veracity, and the value. These concerns are briefly described below.

Data volume - Currently, industries are inundated with voluminous data of all types. The amount of data is growing from terabytes to even petabytes. Tweets from social network sites alone are responsible for up to 12 terabytes of data every day. On an annual basis, power meters contribute to a huge amount of data – 350 billion readings.

Data velocity - Data is being collected at a very fast pace. Sometimes even the delay of a minute may cause a discrepancy in the analyzed output. Especially for time-sensitive systems such as fraud detection, Big Data must be analyzed as it is generated to obtain the best results. For example, 5 million trade transactions per day must be scanned for fraud prevention and exposure.

Data variety - Data can be of any type, regardless of its structured or unstructured nature. Data types include text, audio and video data, sensor data, log files, etc. One of the main reasons that enterprises prefer to work with Big Data is to discover new insights when all such data types are analyzed together. Examples include monitoring live video feeds from surveillance cameras in order to narrow a point of interest and analyzing bulk data growth in videos, documents and images to enhance customer satisfaction.

Data veracity - In the business world, leaders do not completely trust information extracted from Big Data. However, they exploit this information to make better decisions. If someone does not trust the input, she/ he cannot trust the output and will not make decisions because of it. As Big Data grow in size and variety every day, instating confidence in it poses greater challenges.

Data value - This may be considered to be the most valuable aspect, needing careful, first hand attention. Regardless of the arena, enormous efforts are invested in the curation of Big Data with the expectation that they will be content-rich and valuable. However, the richness of the data may be highly valuable for certain communities and may be of no use for others that do not see an applicable outcome from the extraction. Therefore, in the latter case, the investment of effort does not warrant any return. Thus, a primary careful analysis of the value of the Big Data before its curation may help building a competitive advantage.

“Big Data” (White, 2012; Robert, 2012) has today evolved as a predominant research area in which researchers want like to exploit their data engineering capabilities. The size data collections that make up Big Data is a major challenge today; in 2012, for example, it ranged from a few dozen terabytes to many petabytes in a single data set (Watters, 2010). The continued rapid growth of data has imposed the need of new and effective knowledge discovery and data mining (KDD) methodologies as raw Big Data is not directly usable in existing database management systems. Manual data analysis may be efficient if the total amount of data is relatively small but unacceptable for Big Data. KDD helps to automate system analysis and focuses on methodologies for extracting useful knowledge from data. It also assists users in unearthing hidden facts and relationships in datasets that improve the effectiveness and efficiency of their data analysis.

Intelligent knowledge discovery from Big Data is also a challenge that requires a combination of statistics, databases, pattern recognition, machine learning, data visualization, optimization, and high-performance computing in order to perform advanced analysis. Data oriented scientific research is experiencing a potential transformation with the use of Big Data in which it is able to perform data intensive decision making at a new level of effectiveness.

The U.S. Government has also recently announced a national "Big Data” initiative that aims to develop new core techniques and technologies to extract and use the knowledge derived from collections of large data sets to enhance scientific research and provide innovation in managing, analyzing, visualizing, and extracting useful information from large, diverse, distributed, and heterogeneous datasets (Weiss & Zgorski, 2012). An additional goal is to will increase the understanding of human and social processes and interactions and promote economic growth and improved health and quality of life.

3 BIG DATA MODELS

In this section, we briefly review six data models that are actively being discussed in the Big Data (of petabyte size) arena and, to the best of our knowledge, are considered to be reliable, efficient, and leading data models which are able to cope with today’s data challenges. There are other data models being used for Big Data, but the six selected models represent a diversity of mainstream approaches.

3.1 BigTable

In an introductory paper, Chang et al. (2006) outlined Big Table’s simple data model along with its design and implementation. BigTable, primarily used by Google for important projects, is a distributed storage system designed to manage data of petabyte size that are distributed across several thousand commodity servers that allow further horizontal scaling. At the hardware level, these thousands of commodity servers are grouped under distributed clusters that in turn are connected through a central switch. Each cluster consists of various racks, a rack consisting of several commodity computers that communicate with each other though rack switches. The switches are connected to the central switch to help with inter-rack or inter-cluster communications. BigTable efficiently deals with latency and data size issues. It is a compressed, proprietary system with a high performance storage system and is underpinned by the Google File System (GFS) (Ghemawat et al., 2003) for storage of log and data files, and Google SSTable (String Sorted Table). The SSTable file format stores BigTable data internally. In its internal structure the SSTable offers a persistent, immutable ordered map with keys and values as byte strings with the key as the primary search source. The SSTable is a collection of blocks, typically 64KB, though the size is configurable. Unlike relational databases, SSTable is a sparse, multi-dimensional sorted map, indexed by a row key, column key, and a timestamp. The map stores each value as an uninterrupted array of bytes, for example, (row: string, column: string, time: int64) → string.

Figure 2 depicts the web page storage in BigTable. The reversed URL is the name of the row. The actual contents of the pages are stored in the "contents:" column family. The text of the anchors referencing the page is contained in the "anchor:" column family. In the example, CNN webpages are stored as "com.cnn.www" (the row name). The home page is referenced in other pages, here Sports Illustrated and My-look, and thus, the row is appended by columns "anchor:cnnsi.com" and "anchor:my.look.ca." The contents column has multiple versions, in contrast to other anchor cells. In Figure 2, there exists only one version for an anchor cell whereas the contents column possesses three versions of the same data at timestamps t_3, t_5, and t_6. A BigTable cluster houses numerous tables, which in turn are a set of tablets (approximate size100-200 MB) with initial set size to one, but as the table grows, it fragments into multiple tablets. Tablets are the entities that consist of all the data of a row. Figure 3 is the tablet representation in BigTable.

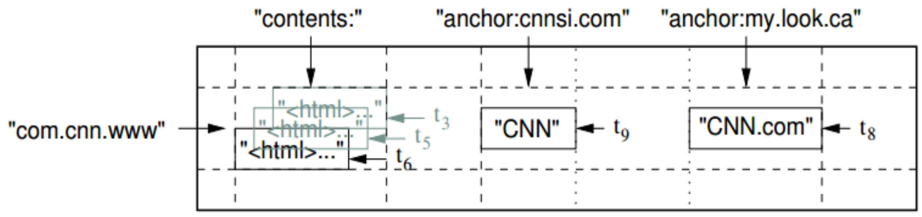


Figure 2. Slice of an example of the BigTable table storing web pages (Source: Chang et al., 2006)

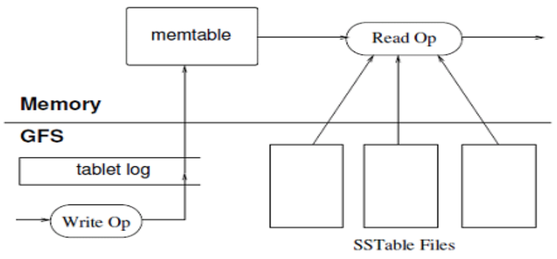


Figure 3. Tablet representation in BigTable table (Source: Chang et al., 2006)

Figure 3 shows the read-write operations in BigTable. As shown, in a write operation, a valid transaction logs into the tablet log. The commit is a group commit to improve the throughput. During the commit moment of the transaction, the data is inserted into the memory table (*memtable*). During the write operation, the size of memtable continues to grow, and once it reaches the threshold, the current memtable freezes and a new memtable is generated. The former is converted into SSTable and finally written into GFS. In a read operation, the SSTables and memtable form an efficient merged view on which the valid read operation is performed.

BigTable relies on Google's in-house, highly-available, loosely-coupled, persistent, distributed locking systems to synchronize accesses to the shared resources, called Chubby (Burrows, 2006). The service was developed to provide coarse-grained locking and offers a reliable distributed file system. The BigTable Data Model is a high availability storage system that is also highly scalable. It is a simple data model and at the lowest end can be imagined as a multidimensional flat file. The technology is leveraged by more than 60 Google products through their APIs.

In the BigTable data model, there is no support for ACID (Atomicity, Consistency, Isolation, and Durability) transactions across row keys. Because the BigTable data model is not relational in nature, it does not support "join" operations nor is there any SQL type language support. This may pose challenges to those users who largely rely on SQL-like languages for data manipulation. In the BigTable data model, at the fundamental hardware level, the inter-rack communication is less efficient than the intra-rack communication, i.e., communication within a rack through rack switches. However, this bottleneck is overcome through specialized software systems. The chubby lock service, used for shared resources synchronization, has limitations on the scalability of the distributed file system.

BigTable is used primarily by Google for its various applications such as Gmail (Copper, 2009), YouTube (Cordes, 2007), Orkut (Whitchcock, 2005), Google Code Hosting, Blogger.com (Google Inc., 2003), Google Earth (Google Inc., 2001), My Search History, Google Book Search (Google Inc., 2004), Google Maps (Whitchcock, 2005), Google Reader (Wetherell, 2005), web Indexing (Chang et al., 2006), and Map Reduce (Chang et al., 2006: p. 3).

3.2 Cassandra

Cassandra, originally devised by Lakshman and Malik (2011), is a distributed, open source data model that handles petabyte amounts of data, distributed across several commodity servers. Cassandra's robust support, high data availability, and no single point of failure are guaranteed by the fact that the data clusters span multiple datacenters. This system offers master-less, peer-peer asynchronous replicas and is a member of the NoSQL revolution. Peer-peer clustering (the same role for each node) helps achieve low latency, high throughput, and no single point failure. In the data model, rows are partitioned into tables. The very first column of the table forms the primary key, also known as the partition key. A row is clustered by the rest of the columns. As Cassandra is a member of NoSQL family, it does not support joins and sub-query operations, but it encourages denormalization for high data availability. The Cassandra data model implements parallel data operations resulting in high throughput and low latency and also adopts a data replication strategy to ensure high availability for writing. Unlike relational data models, Cassandra duplicates data on multiple nodes to help ensure reliability and fault tolerance.

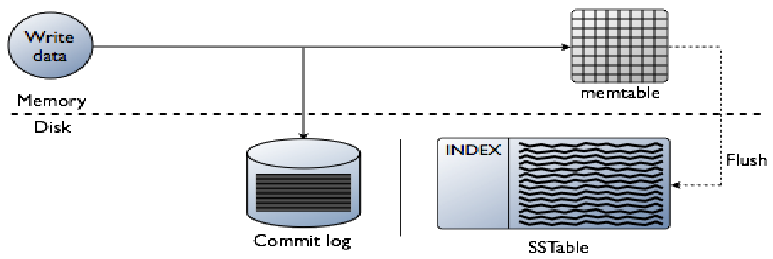
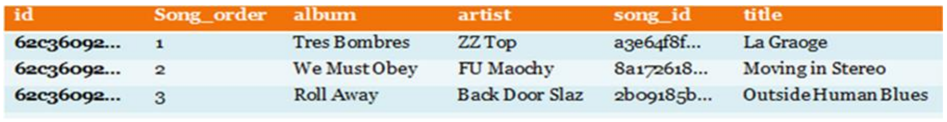


Figure 4. Writing path in Cassandra (Source: DATASTAX documentation, Apache Cassandra^{TM} 2.0)

Figure 4 shows the writing path in Cassandra. Whenever a write operation occurs, the data are stored in the form of a structure in the memory; in Cassandra terminology this is called a *memtable*. The same data are also appended to the commit log on the hard disk, which provides configurable durability. Each and every write on a Cassandra node is also received by the commit log, and these durable writes are permanent and survive even after hardware failure. Setting the durable write to true is advisable to avoid the risk of losing data. Though customizable, the correct amount of memory is dynamically allocated by the Cassandra system, and a configurable threshold is set. During the writing operation if the contents exceed the limit of memtable, the data including indexes, are queued, ready to be flushed to SSTables (Sorted String Table) on a disk through sequential I/O. The Cassandra data model comes equipped with a SQL-like language called Cassandra Query Language (CQL) to query data. Figure 5 depicts a sample set of CQL queries and the sample output resulting from executing the select query.

```
Example query 1:
        CREATE TABLE playlists(
        Id uuid,
        Song_order int,
        Song_id uuid,
        title text,
        album text,
        artist text,
        PRIMARY KEY (id, song_order));

Example query 2:
        INSERT INTO playlist (id, song_order, song_id, title, artist, album)
          VALUES (62c36092-82a1-3aoo-93d1-46196ee77204, 4,
                           7db1a490-5878-11e2-bcfd-o8oo2ooc9a66,
                           'ojo Rojo', 'Fu Manchu', 'No One Rides for Free');
Example query 3:
        SELECT * FROM playlists;
```



id	Song_order	album	artist	song_id	title
62c36092...	1	Tres Bombres	ZZ Top	a3e64f8f...	La Graoge
62c36092...	2	We Must Obey	FU Maochy	8a172618...	Moving in Stereo
62c36092...	3	Roll Away	Back Door Slaz	2b09185b...	Outside Human Blues

Figure 5. Sample CQL query set and sample output (Source: DATASTAX documentation, CQL for Cassandra 1.2)

Literature suggests that Cassandra does not have a PHP driver that can be used to program a system. There are some drivers that do not have proper documentation on how to create a PHP extension for a window application. There is an increasingly growing list of prominent users that rely on the computational capabilities of Cassandra for their data management partially or fully, for example, Facebook, IBM, Netfix, Rackspace, Twitter, etc.

3.3 HBase

HBase, written in Java, is a non-relational, column-oriented, open source, and distributed database management system (Dimiduk & Khurana, 2012) that uses Hadoop Distributed File System (HDFS). It is designed to deal with very large sparse datasets, is a member of the NoSQL family, and follows the master-slave concept as well. HBase is not a relational database, however, and does not support SQL; instead, it is a column-oriented data model that consists of sets of tables, consisting of rows and columns. Similar to a relational database, each table, has a primary key that is used to access the data. HBase has the ability to group many columns (attributes) together into what are called column families, with the elements of a column family are all stored together. A new column can be added to a column family at any time.

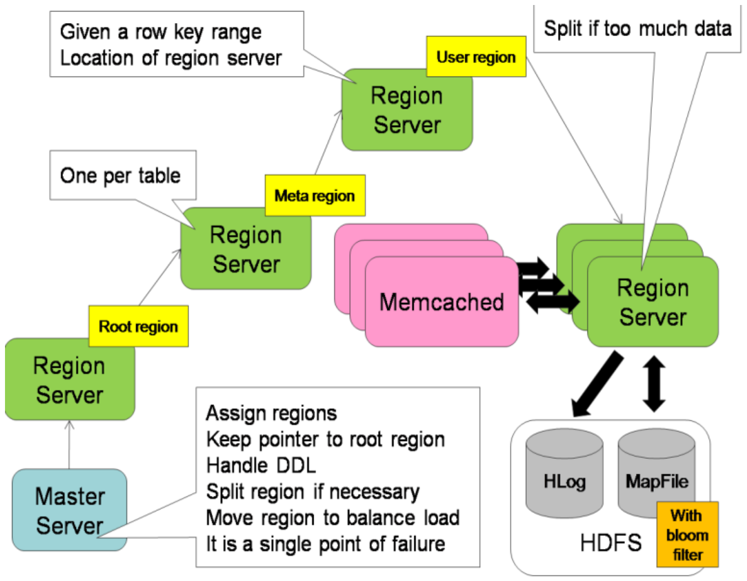


Figure 6. HBase architecture (Source: Ho, 2010)

Figure 6 illustrates the architecture of the HBase data model. Although the use of HDFS underneath absorbs data replication, consistency, and resiliency burden from HBase, it creates extra network latency to the DB server (region server) and HDFS data nodes. As an abstraction, the data are stored at the region servers. Before any database operation is performed, the required region server must be located first. The key-to-server mapping, stored in a simple table, helps locate a required server. The client contacts the Master server that then gives information about the region server that holds the "Root Region" table. Based upon the mapping, this Root Region server directs the client to another region server that holds the "Meta Region" table, which consists of the mappings of user tables to region servers. The Meta Region server looks up the meta-region table and tells the client to contact a particular region server. That server holds the "User Region" table that has the mapping of key range to region server. The client contacts the User Region server with a row key, and the server finally redirects it to the server that holds the required data. The system caches the results, which may be used in the future, thus improving system performance. In HBase terminology, the Master server is called a heavy-duty worker node because it monitors the activities of all the region servers and coordinates with them. Although the Master server is a special machine, it is vulnerable to a single point of failure. Unlike in Cassandra, in HBase the Memcached implements the in-memory data storage, and on-disk data (including logs) are stored in HDFS files stationed in Data Node servers. Recently, HBase has attained wide acceptance in various big industries such as Yahoo! Inc. and Facebook.

3.3.1 HDFS

The Hadoop Distributed File System (HDFS), written in Java programming language, is a highly fault-tolerant file system primarily aimed at deploying and running on low-cost commodity hardware (Borthakur, 2007). It is uniquely structured for complex applications that deal with large datasets (up to terabytes) and streaming access with high throughput and scale to hundreds of nodes in a single cluster. HDFS believes in the principle that in case of an extremely large dataset, moving computation is cheaper than moving data. Moving the requested computation, which is relatively much smaller than the data it is required to operate on, to execute near the data greatly minimizes network congestion and thus enhances the throughput of the entire system.

Figure 7 depicts the complex architecture of HDFS. The HDFS is easily portable to different platforms and has master-slave architecture. As shown, a typical HDFS cluster consists of a master server called NameNode and a number of slaves called DataNodes, placed in racks. The NameNode assists in accessing files by the clients and also manages file system namespace operations, such as opening, closing, and renaming files and directories. It also is responsible for blocks mapping to DataNodes. The DataNode manages the data storage in files and takes care of read/write requests from the client of the file system. A file in HDFS is split into blocks, and a DataNode is responsible for their creation, deletion, and replication at the request of the NameNode, which is the repository for all HDFS metadata. In a typical cluster, the NameNode is the decision maker for the block replications and receives a Heartbeat and a Blockreport from each individual DataNode at regular periods. The Heartbeat indicates the DataNode's proper functioning whereas the Blockreport reports about all the blocks on that particular DataNode. The HDFS reliably stores large files across multiple machines in a typically large cluster, and each file is stored as a sequence of blocks, which in turn are replicated across other machines for the fault tolerance. Block size, replication factor, and number of replicas are configurable attributes.

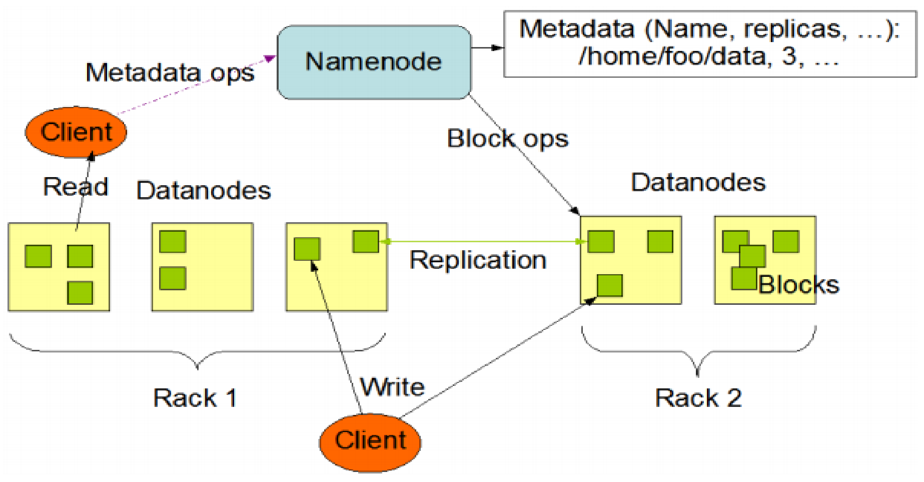


Figure 7. HDFS architecture (Source: Hadoop 0.18 documentation, The Hadoop Distributed File System: Architecture and Design)

HDFS is used as a bare base file system by HBase (Dimiduk & Khurana, 2012). Although by itself it is not used extensively, combined with other components in a complex package, known as Hadoop, its potential is enormous. Yahoo! Inc., Facebook, IBM, TIBCO, MetaScale, and MapR Technoloies Inc. are some of the vendors that either use or support it.

3.3.2 MapReduce

MapReduce is a programming model that has recently attained momentum in the Big Data community (Ranger et al., 2007). It exploits parallel and distributed algorithms underlying a cluster to process efficiently very large data and uses master-slave architecture underneath. As its name portrays, it is a tight conglomeration of two functions, called Map and Reduce. The model runs various smaller tasks in parallel on multiple distributed servers. As a complete framework, the MapReduce model manages data transfers and communication among different components of a system and takes care of redundancy and fault tolerance challenges. The libraries of MapReduce are open-source and available in multiple languages; today, its most popular implementation is Apache Hadoop (Borthakur, 2007). The MapReduce framework is very efficient in processing very big data using multiple commodity hardware machines within a cluster. The localness of data is advantageous here reducing the need to send large amounts of data across multiple machines. The processing threads are migrated to the data units to process the data near the storage unit. The processing units are much smaller in size than the data units, thus comparatively decreasing data transmission, even in longer movements. This in turn reduces the overall time and the bandwidth requirements. In the Map function, the master node accepts the complex problem as input, slices that into smaller sub-problems, which are eventually delegated to the worker (slaves) nodes that run them on localized data units and send the output back to the master node as a key-value pair. In the Reduce function, the master node collects all key-value pairs and synthesizes them into a problem-specific way, forming a whole that forms the output for the original problem.

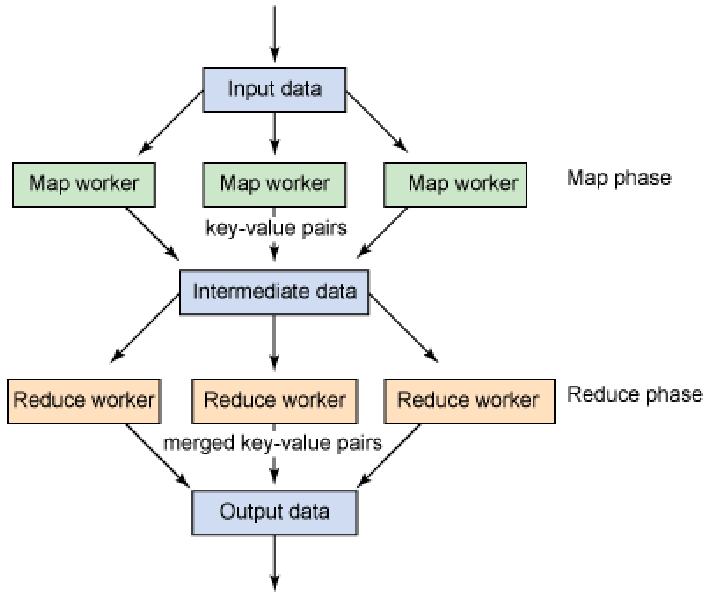


Figure 8. Simple architecture of the MapReduce framework

Let us try to understand the MapReduce concept further with the help of an example.

```
Example query 1: Write the Map and Reduce functions that count the appearance of every word in a
set of documents.

               function map(String name, String document):
                 // name: name of the document
                 // document: contents in the document
                 for each word wo in document:
                   outputFunc (wo, 1)

               function reduce(String word, Iterator wordCounts):
                 // word: a word
                 // wordCounts: a list of aggregated word counts
                 TotalCount = 0
                 for each wc in wordCounts:
                   TotalCount += ParseInt(wc)
                 outputFunc (word, TotalCount)
```

It can be noticed that both functions have key-value pairs as their input parameters. The role of the key in the Map function is to identify the data that are to be processed by the associated map instance. There may be several instances of the Map function poured across multiple machines. The Map function returns the output in the form of the key-value pairs, which are used by the Reduce function to reduce this large set of key-value pairs to a single output, possessing the same input key as the final result of the original complex problem. MapReduce is extremely useful for a large pool of computing applications such as Google's index of the World Wide Web, distributed sorting, web access log stats, pattern-based searching, etc.

3.4 MongoDB

MongoDB (from the word humongous) is an open source, document-oriented, NoSQL database that has lately attained a presence in the data industry (Chodorow & Dirolf, 2010). It is considered to be one of the most popular NoSQL databases competing today and favors master-slave replication. The role of the master is to perform reads and writes; whereas the slave is confined to copying data received from the master, performing read operation, and backing up data. The slaves do not participate in write operations but may select an alternate master in case of failure of the current master. Underlying MongoDB is the binary format of JSON (Java Script Object Nation)-like documents. It uses dynamic schemas, unlike traditional relational databases. The query system of MongoDB can return specific fields. The query set compass searches by fields, range queries, regular expression search, etc. and may include user-defined complex JavaScript functions. MongoDB takes advantage of flexible schemas and the document structure in a grouping, called Collection, may vary. Common fields of various documents in a collection can have disparate types of the data.

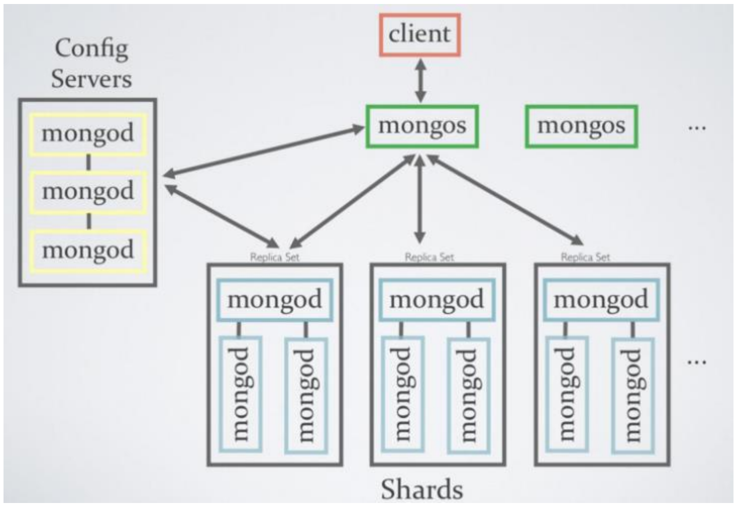


Figure 9. MongoDB architecture (Source:MongoDB 1.6 Adds Sharding and Replica Sets, by Abel Avram, infoq.com)

Figure 9 is the updated architecture of MongoDB. The latest release of the data model is equipped with more advanced concepts such as sharding that address the scaling issues while the replica sets assist in automatic failover and recovery. Sharding is a process that addresses the demand of the data growth by storing data records across multiple machines without any downtime. Each shard (server) functions as an independent database and all the shards collectively form a single logical database. Each config server possesses the metadata about the cluster such as information about the shard and data chunks on the shard. For protection and safety, multiple config servers exist, and if one goes down, another takes charge without any interruption in shard functioning. Mongod is the key database process and represents one shard. In a replica set, one of the mongod processes serves as the master, and if it goes down, another is delegated for the master's responsibilities. Mongos intermediates as the routing process between client and sharded database.

The MongoDB data model is equipped with suitable drivers for most of programming languages used to develop the customized systems that use MongoDB as their backend player. There is an increasing demand to use MongoDB as a pure in-memory database; in such cases, the application dataset will always be small. Although it is probably easier for maintenance and less taxing for a database developer, this can be a bottleneck for complex applications that require large scale database management capabilities.

Some of prominent users of MongoDB are MetLife, Craigslist, Forbes, The New York Times, Sourceforge, and eBay. For example, The New York Times has its own form-building application that allows photo submission. This application is completely backed up by MongoDB. Sourceforge, on the other hand, has shown more confidence in MongoDB and uses it to store back-end pages. eBay uses it for search suggestion applications as well as for its own internal Cloud Manager.

3.5 CouchDB

CouchDB (Cluster of Unreliable Commodity Hardware Data Base) is an open source, NoSQL, web inclined database that uses JSON to store the data and relies on JavaScript as its query language (Brown, 2011). It uses HTTP for data access as well as multi-master replication. CouchDB does not use tables at all for data storage or relationship management; rather the database is a collection of independent documents, maintaining their self-contained schemas and data. In CouchDB, multi-version concurrency control helps avoid locks during write operations while the document metadata holds the revision information. Off-line data updates are merged and stale ones are deleted.

Figure 10 is a simplified architecture of a two nodes cluster of CouchDB. The figure shows a lightweight webserver, NGINX. This is the front face of the entire cluster and serves as a reverse proxy here. It accepts incoming requests from the client and forwards them to the CouchDB, specified in the configuration settings. CouchDB is equipped with a built-in administration web interface, called Futon (Anderson, Slater, & Lehnardt, 2009).

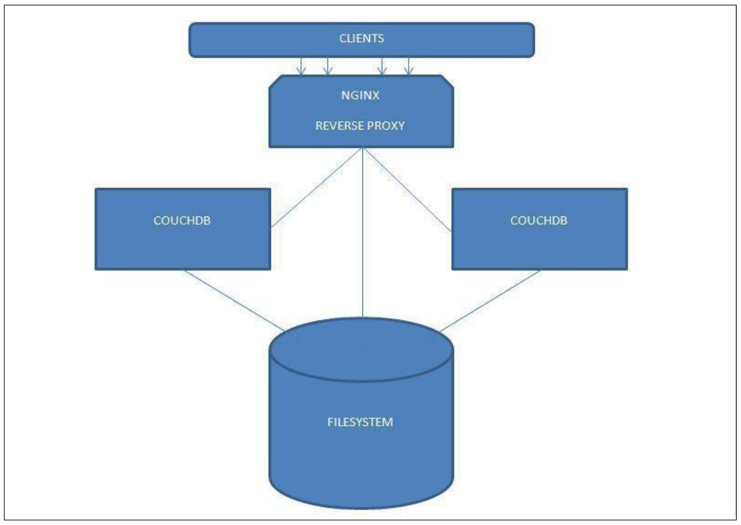


Figure 10. CouchDB simple architecture (Source: A Guide to CouchDB Installation, Configuration and Monitoring, Zhirayr, 2012)

Figure 11 is the welcome screen of Futon that can be loaded using an HTTP access such as http://127.0.0.1:5984/_utils/. As stated already, APIs are used to access the data in CouchDB. A few of the example queries are expressed here using the command-line utility curl. By default, it issues GET requests only; however, POST requests can be issued by explicit request.

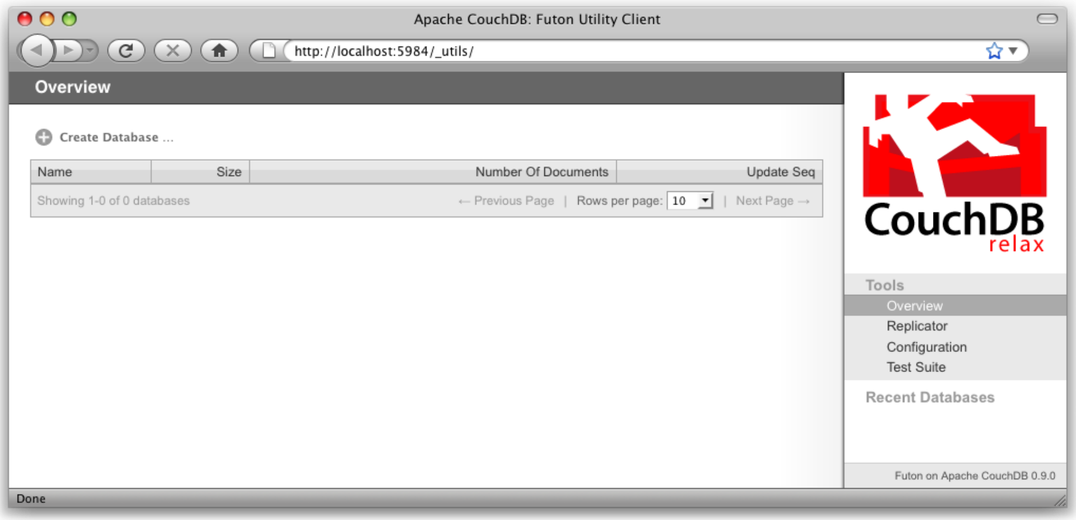


Figure 11. CouchDB web interface, Futon (Source: CouchDB, The Definite Guide, couchdb.org)

The usability of CouchDB is growing; prominent users include Ubuntu, The British Broadcasting Corporation (BBC), Credit Suisse, Meebo, etc. Ubuntu used it for its in-house synchronization service but relinquished its use in 2011. BBC exploits the computational capacity of CouchDB for its platforms used for dynamic content. Credit Suisse is using CouchDB for its internal commodity departments.

3.6 CrowdDB

CrowdDB (Franklin et al., 2011) can be understood as a database system, extended to take advantage of human knowledge that is designed to answer any query. Data volume is not the only challenge of Big Data analytics; enormous variety, ambiguity, and incompleteness in data as well as in queries are also potential issues. Scientific and technical advances in scalable data processing suitably tackle the volume issues of the data, but there are several data-centric tasks where human involvement activity and intelligence are more useful than modern algorithms. In situations where it is either almost impossible or too expansive to correctly perform a task for a traditional database, human involvement can save the day. For such situations, Crowdsourcing is a major problem-solving and data-gathering paradigm.

CrowdDB harnesses human computation to handle such situations by employing a micro-task crowdsourcing platform such as Amazon's Mechanical Turk (AMT) (http://www.mturk.com). This type of platform facilitates a mechanism by which large numbers (e.g. a million) people can participate in paid work through the internet. Typically, AMT can be used for such tasks that are easier for people to complete than for computers, e.g., writing a query to discover the best of a collection of images that can be used in a motivational slide show. Although a query can be written, there is no efficient way in relational database management systems (RDBMS) to answer the query reliably and correctly until a stored corpus of picture relevancy vs. topic exists. Rather, this computer unanswerable query can be better answered by people's judgment, especially through the experts (crowd) of the related field having internet access. CrowdDB recognizes this need and appends the crowd functionality to the database management system (DBMS). The relational query engine expands to a small number of new operators that solicit human input through creating and submitting task requests to a micro-task crowdsourcing platform. Thus, collectively, human and machine function as query processing elements, although they differ fundamentally in the way they process.

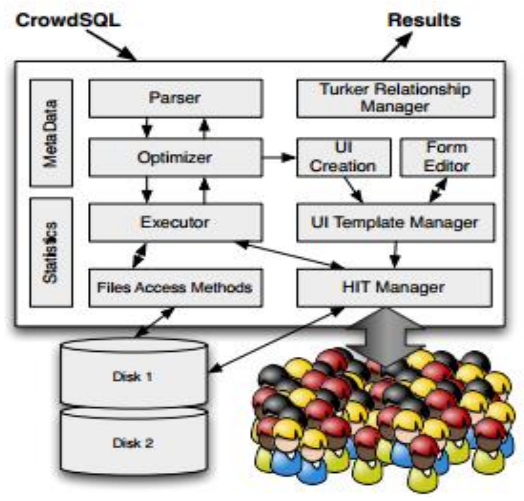


Figure 12. CrowdDB architecture (Source: AMPed at UC Berkeley, by Steve Todd, EMC Innovation Network)

For example, in a given context, the terms I.B.M, IBM, or "International Business Machines" refer to the same entity, but a database system (computer) considers them three different entities, and their mistaken assignment for one another will return erroneous results. On the hand, a human is capable of judging that all the three items refer to a single entity, and this will help produce correct answers. Similarly, in selecting the best images for a particular representation, human expertise produces better results than a computer. Thus, in the CrowdDB data model, the crowdsourcing platform is the key component.

A crowdsourcing platform generates a marketplace where the requester assigns tasks, and based on their expertise and preferences, including the price of the task, the workers agree to execute the tasks. AMT is the leading platform offering APIs such as *createHIT (title, description ...)* that facilitate the entire process and support micro-tasks. A micro-task generally does not require any special training sessions and takes a minuscule amount of time to complete, except for extreme cases that may take up to one hour. In the AMT platform exclusively, the requester also specifies a price/reward (minimum $0.01) along with the task that, on satisfactory task completion, will be given to the associated worker. As of now, the payment mode is restricted to a U.S.-based credit card.

Figure 12 is the general architecture of CrowdDB. The CrowdDB data model has its own query language, CrowdSQL, which can be seen as a moderate extension of SQL although it supports crowdsourcing. Some interesting CrowdSQLs are discussed later in this section. As Figure 12 illustrates, the query is fired using CrowdSQL. In this model, the applications can be built in the usual traditional way; the underlying complexities in dealing with crowdsources are taken care by CrowdDB itself. CrowdDB accesses the local tables as its first attempt to answer queries, and then it invokes the crowd otherwise if there is no answer, storing the crowd-fetched results in the database for future requests. In the figure, the left-side is the stack of usual components of a traditional data model, which are extended here to facilitate data management from crowdsourcing. The component stack on the right side of the figure is the interface, responsible for active interaction with crowdsourcing platform. The top component, Turker Relationship Manager, performs requester's duties that involve approving or rejecting assignments, payment, etc. The HIT Manager mediates the interaction between CrowdDB and the crowdsourcing platform. It also interacts with the storage engine to put the crowd-obtained data into storage. To have a full understating of CrowdDB architecture, readers are encouraged to consult the related reference (Franklin et al., 2011).

Traditionally, CrowdDB is not a Big Data model and has been considered as a kind of relational data model, supported by the power of human computation. What motivates us to consider CrowdDB as a Big Data model is its crowdsourcing nature, which resolves many complex tasks that are beyond the computational power of traditional database management systems. The inclusion of the crowdsource tags in CrowdSQL contributes to the richness of language and aids in dealing with scaled up data.

CrowdSQL: As mentioned above, CrowdSQL is a minimal extension of SQL that is able to support crowdsourcing. CrowdSQL queries are written in the same way as the traditional database queries, and most of the time the user is unaware of the fact that the query code involves crowdsourcing.

In CrowdSQL, the keyword CROWD marked before any entity (e.g., column, table, etc.) informs the query engine to explore the crowdsources for that particular entity. The SQL equivalent default value for a CROWD column is CNULL, and only those columns initialized by CNULL will be crowdsourced; otherwise, they will be treated as normal SQL columns, initialized with NULL. This sometimes causes side-effects to the queries. A prototype of CrowdDB has been implemented at the research level. Although it is not yet advertised commercially, the vendors are eagerly waiting for its first stable commercial release.

4 DISCUSSION

Important features of the reviewed data models are shown and compared in Table 1. These model have been chosen as because of their capability to best address the challenges posed by the transformation of data into Big Data. This transformation has high potential for science research, in particular Big Science, and also eloquently converted the operational-oriented definition of Big Science into a data-driven definition. Although the list of the models reviewed is limited in size, each item in the list is robust and self-contained, is being considered for use in Big Science, and has left a mark on the modern data world. In Figure 13 we have tried to rank some of the models discussed in this paper. Our calculation of popularity is based on the mechanism used by solidIT (2014), which counts the usage of the name of the model on various web spaces to derive a popularity ranking, for example, search frequency in Google Trends, frequency of model name tweeted in Twitter, etc. Unfortunately, the popularity data for BigTable and CrowdDB is not available with solidIT.

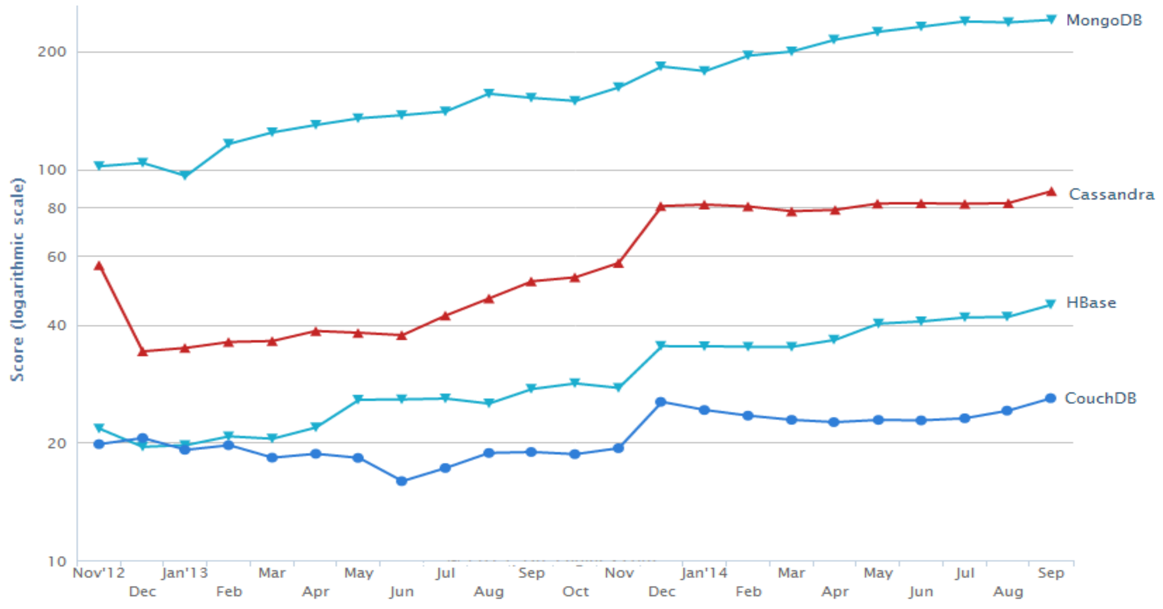


Figure 13. Popularity trends of some of the Big Data models (Source: solidIT, 2014)

Of the models shown in Table 1, CrowdDB arguably is not a Big Data model per se (Franklin, 2012). It is discussed here intentionally and strategically along with the others because its crowdsourcing nature can resolve many complex tasks that are beyond the computational power of traditional database management systems. The inclusion of crowdsource tags in CrowdSQL contributes to the richness of language and aids in dealing with the scaled up data. Furthermore, unlike other traditional Big Data models, MapReduce (Section 3.3.2) is a software programming model that is known for its efficient parallel and distributed computing capabilities. It becomes extraordinarily useful when it is integrated with other data models to further enhance the parallellism in data processing. As of late, the interweaving proximity of the MapReduce programming model with the Apache Hadoop framework has drawn significant attention and popularity in the recent buzz of Big Data.

The importance of indexing (Bertino et al., 2012) in databases has been proven, particularly when data size is extremely large. It improves searching for desired data in large tables and helps in quickly locating data through bypassing the traversal of each and every row. Indexes support efficient execution of more often used queries, especially for read operations. An index may include a column or more, and sometimes the size of an index may grow larger than the table it is being created for, but eventually the index will provide a rapid lookup and fast access to the data. This compensates for the overhead of having indexes.

Indexing in Big Data is a challenge for several reasons, primarily because the volume and velocity of Big Data are very large; a generated index should be small (a fraction of the original data) and fast in comparison with the data because searches must consider billions or even trillions of data values in seconds. As Big Data models entertain multi-variable queries, indexing is required to be in place for all in-operation variables to provide faster access for individual variable search results that are further combined for a collective response as a whole. Models should have the ability to produce smaller indexes when there is a possibility for granularity reduction. The indexes should be able to be portioned easily into sections whenever the opportunities for parallel processing arise. As the rate of data generation under the Big Data era is very large, in order to process data in place, an index should be built at the same rate. The Big Data models discussed in this paper deal with the indexing issues in their own ways and are briefly explained here.

In MongoDB, indexes (Chodorow& Dirolf, 2010) that use a B-tree data structure are generated at collection (equivalent to a table in relation databases) level. To support a variety of data and queries, there are different types of indexes available: 1) Default_id - this exists by default on the _id field of collections and also serves as a unique id; 2) Single Field - this is a user defined index on a single field of a document; 3) Compound Index - this is a user defined index on multiple fields; 4) Multikey Index - this is used to index a field that holds an array of values; 5) Geospatial Index - this is a very special index that MongoDB (unlike other models) provides to support geospatial queries and has two types: (i) 2d spatial index for planer geometry queries, (ii) 2sphere spatial index for spherical geometry queries; 6) Text Index - this is the index type that helps search for the string or text content in a collection; and 7) Hashed Index- this helps index the hash of the values of a field.

In Cassandra, similar to a relational database, the first index is the primary key that ensures the uniqueness of each record. The primary index, applied to a column family, serves as the index for its row keys and is maintained by each node for its associated data (Lakshman, & Malik, 2011). Cassandra also provides secondary indexes on column values that can act as an additional filter to the result set. Although Cassandra has indexing capability, it does fall short before MongoDB, which offers a wide range of indexing.

CouchDB on the other hand does not infuse the power of indexing directly, unlike relational database models. In contrast, the usage of index in CouchDB is assisted by the Map and Reduce functions of the predefined MapReduce framework; any variations in the document structure of the data are well absorbed by these functions and encourage parallelism and independent computation in indexes. The indexes created by map functions limit the window of the targeted data; the reduce queries are allowed to run to seek optimal efficiency. Therefore, while writing the map function, building an index should also be a concern. In CouchDB, the amalgamation of a map and a reduce function is termed as a view.

HBase, similar to CouchDB, is also not equipped with robust indexing capability (Dimiduk & Khurana, 2012). Secondary indexes are not built in by default but can be generated in other tables that may require periodic updating, performed by MapReduce. Another possibility is to write to the index table while writing to the data table. Although each approach has both merits and demerits, all in all HBase does not aggressively support indexing.

BigTable data models such as HBase are also not good promoters of indexing (Jin et al., 2011). It has already been stated that technically BigTable is a map that is sparse, distributed, persistent, multidimensional, and sorted, although the map is indexed by a row key, column key, and a timestamp. As CrowdDB is partially (excluding the human involvement aspect) similar to a traditional relation database, it has all the basic indexing techniques of any relational database model. In sum, if we compare the indexing richness in all the reviewed Big Data models, MongoDB undoubtedly is the much preferred choice.

Versioning (Shaughnessy, 2012), also called version control or revision control, is another important feature in databases. As changes in any database obviously happen and are expected to be frequent, a mechanism was devised to identify the changes, called versioning. New changes are assigned a number, known as the version or revision number. Versioning allows finding the latest or any particular version from database tables and also significantly eases audit trailing of the data in the systems. There is some basic information that is required in general for audit trailing - who edited the data, what changes have been made, time the change occurred, etc.

MongoDB versions the document at the application layer although this is not a native feature. One way suggested to achieve efficiency is to have two different databases, one the application data store and the other the audit trail. CouchDB employs the Multi Versioning Concurrency Control (MVCC) (Suryavanshi & Yadav, 2012; Cattell, 2011) system to address the versioning issues. Despite this, CouchDB does not warrant default document versioning support. It does implement a form of MVCC, especially to preclude the strict requirements of exclusive locking on data files while performing the write operation. Version conflicts can be resolved at the application level by merging the conflicted data into one file, followed by the eradication of the rancid part.

Table 1. Feature comparison of the reviewed data models

Model / Features	**BigTable (2004)**[1]	**Cassandra (2007)**	**HBase (2007)**	**MongoDB (2007)**	**CouchDB (2005)**	**CrowdDB (2006)**
Type	Key-value pair	Hybrid of Key-value pair & Tabular	Column-oriented (Key-value pair)	Document-oriented (JSON format)	Document-oriented (JSON format)	Tabular
Relational nature	No	Yes	No	No	No	Yes
Developer	Google Inc.	Facebook Inc. (Avinash Lakshman, Prashant Malik)	Microsoft (Powerset)	10gen	IBM (Damien Katz)	Jeff Howe
Language of development	C, C++	Java	Java	C++	Erlang	Html, JavaScript, Java
Query language	APIs in C++	CQL	1.Pig latin 2. HQL	MongoDB ad-hoc query language	JavaScript	CrowdSQL
Query nature (SQL based)	No	Yes	1.No 2.Yes	No	No	Yes
High availability	Yes	Yes	Yes	Yes	Yes	No

[1] Year of invention

High scalability	Yes	Yes	Yes	Yes	Yes	No
Single point of failure	N/A	No	Yes	No	No	Yes
Open source	No	Yes	Yes	Yes	Yes	No
Versioning	Yes (built-in timestamp)	Yes (External timestamp at query level, but no built-in)	Yes (built-in timestamp)	Yes (different database for audit trails, not native)	Yes (MVCC)	No (Except basic versioning of RDBMS)
Indexing	Basic on map, supported by timestamps	Basic primary and secondary	Basic (Secondary not by default)	Advance and wide variety (includes spatial indexing)	Basic associated map and reduce functions	Basic as in relational data models
Data Processing nature	Batch processing	Streaming and atomic batches	Batch processing	Batch processing and event streaming	Batch processing	Batch processing

In HBase, version management of the data is assisted by timestamps. A change at a particular point of time is updated into the database along with the new timestamp as a new version. In HBase, all columns are eligible to have any number of versions, and accessing data at a particular given time or any time range is well facilitated although the approach contains some unfavorable issues also. Similarly Google's BigTable addresses versioning with the help of timestamps. Each cell in a BigTable is capable of accommodating multiple versions of the same data, which are tagged by timestamps (64-bit integers); the versions are sorted in decreasing order of timestamp, providing the most recent version in the front of the queue to be read first. If assigned by BigTable, a tag depicts the real time in microseconds; otherwise it can be crafted explicitly by the applications that themselves generate unique timestamps to avoid any collision.

In Cassandra, however, there is no built-in mechanism to handle data versioning, but timestamps can be used at the query level to attribute versions. The Cassandra Query Language (CQL) strictly follows the guidelines of semantic versioning (OSGi Alliance, 2010) in version assignment and tags the version in the X.Y.Z format, where X, Y, and Z are integers, reflecting major, minor, and patch level values. In the case of CrowdDB, except for the basic versioning at the RDBMS level, the versioning of CrowdDB as a whole is an open question that the scientific community needs to address. Based on the discussion, we advise using CouchDB if an eloquent versioning of the data is an important consideration in Big Data management.

The style of data processing is another important aspect discussed in this paper. Batch processing (Damodaran & Vélez-Gallego, 2012) and real-time data processing (streaming) (Gulisano et al., 2010) have their own merits and demerits. The former is efficient when a large volume of data needs to be processed. The transactions are collected in groups and processed over a period of time. Data are collected, entered, processed, and finally batch results are generated. In contrast, the latter is more desirable when processing involves continual input and output, and data are processed over a small period of time (near real time). Stream processing is efficient in those situations in which input data must be processed without being stored.

Google's BigTable has innate support for batch processing. Similarly, the HBase original use case supports batch processing of the data although over the time, several packages or frameworks have surfaced that claim to stream data when integrated with HBase. Cassandra, on the other hand, primarily strongly advocates data streaming although it also allows atomic batches. Using batches offer better transactional integrity in the case if the coordinator succumbs mid-batch. An atomic batch promises the success of all the batches if any part of the batch succeeds. MongoDB favors batch processing for bulk inserts and updates. It also provides event streaming if a system generates events in event storage. Similarly, CouchDB also performs batch processing for importing, updating, or deleting the bulk amount of data. CrowdDB also executes batch processing as crowdsourcing is performed in batches (Ramesh, 2011). The implementation of batch processing uses multiple queues that are designed to be populated by the CrowdDB operators.

We intend to continue the inter-comparison discussions on the data models mentioned in this paper with other comprehensive features. One such potential comparison is to benchmark the performance metrics for each of the data models discussed here, by employing a common testing dataset. We believe that performance benchmarking of Big Data models is a challenging task that is certainly beyond the scope of this brief review paper but will be of future interest.

5 LIMITATIONS

This paper briefly reviews six prominent modern Big Data models. To the best of our knowledge, these models are on the center stage of the discussions and usages in research, academia and industry for handling Big Data challenges. The selection of only these six out of the comprehensive pool of Big Data models, however, can be a debatable issue. A reader may find the selection and the size of the selected cluster of the data models to be a limitation of this paper. When it comes to the discussion of a small fraction of the existing Big Data models, there are many different attributes that carry equal importance, weight, and rationale for comparison. The features chosen for Table 1 are in our opinion among the most significant and provide a meaningful comparison among these Big Data approaches.

6 CONCLUSIONS

The increasingly rapid growth of large, complex, heterogeneous, and multi-dimensional data in recent years has raised concerns about the capacity of the existing data management software and hardware infrastructure. There are many different types of data sources, such as sensors, scientific instruments, and the Internet (especially the social media), that contribute to the data proliferation. The comparatively slower development of novel and efficient data models to process complex and large-scale data poses great challenges that require immediate attention from academia, research, and industry. In this paper, we have reviewed six leading data models of the modern NoSQL era that, to the best of our knowledge, are able to process Big Data adequately up to the petabyte range. The data models were discussed and compared as to a number of important features. We hope this paper will be a helpful introduction to readers interested in Big Data.

7 REFERENCES

Anderson, J. C., Slater, N., & Lehnardt, J. (2009) CouchDB: The Definitive Guide (1st ed.), *O'Reilly Media*, p 300. ISBN 0-596-15816-5.

Bertino, E., Beng, C. O., Ron, S.D., Kian, L.T., Justin, Z., Boris, S., & Daniele, A. (2012) Indexing techniques for advanced database systems. Springer Publishing Company, Incorporated.

Borthakur, D. (2007) The Hadoop Distributed File System: Architecture and Design. *Hadoop, The Apache Software Foundation*. https://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf

Brown, M.C. (2011) Getting Started with CouchDB (1st ed.), *O'Reilly Media*, p 50. ISBN 1-4493-0755-8.

Burrows, M. (2006) The Chubby lock service for loosely coupled distributed systems. *In Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pp 335-350, USENIX Association: Berkeley, CA .

Carino, F. & Sterling, W. M. (1998) Method and apparatus for extending existing database management system for new data types. U.S. Patent No. 5,794,250.

Cattell, R. (2011) Scalable SQL and NoSQL data stores. *ACM SIGMOD Record 39*(4), pp 12-27.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2006) Bigtable: a distributed storage system for structured data. *In Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation 7*, Seattle, WA.

Chodorow, K. & Dirolf, M. (2010) MongoDB: The Definitive Guide (1st ed.), *O'Reilly Media*, p 216. ISBN 978-1-4493-8156-1.

Copper, J. (2009) How Entities and Indexes are Stored. *Google App Engine, Google Code*.

Cordes, K. (2007) YouTube Scalability (talk). *Their new solution for thumbnails is to use Google's BigTable, which provides high performance for a large number of rows, fault tolerance, caching, etc. This is a nice (and rare?) example of actual synergy in an acquisition*.

Damodaran, P. & Vélez-Gallego, M. C. (2012) A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications 39*(1), pp 1451-1458.

Demiryurek, U., Banaei-Kashani, F., & Shahbi, C. (2010) TransDec: A spatiotemporal query processing framework for transportation systems. *26th International Conference on Data Engineering*, Long Beach, California, USA, pp 1197 -1200.

Dimiduk, N. & Khurana, A. (2012) HBase in Action (1st ed.) *Manning Publications,* p 350. ISBN 978-1617290527.

Franklin, M. J. (2012) CrowdDB: Query Processing with People and Machines. *New England Database Society*, *HP/Vertica Computer Science Lounge*, Brandeis University, USA.

Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S., & Xin, R. (2011) CrowdDB: answering queries with crowdsourcing. *Proceedings of the 2011 International Conference on Management of Data*, Athens, Greece. [doi>10.1145/1989323.1989331]

Furner, J. (2003) Little Book, Big Book: Before and After Little Science, Big Science: A Review Article, Part I. *Journal of Librarianship and Information Science 35*(2), pp 115–125. doi:10.1177/0961000603352006.

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003) The Google File System. *19th Symposium on Operating Systems Principles,* Lake George, NY: The Association for Computing Machinery.

Google Inc. (2001) Google Earth. Retrieved from the World Wide Web September 11, 2013: http://www.google.com/earth/

Google Inc. (2003. The Story of Blogger. Blogger.com. Retrieved from the World Wide Web: http://www.blogger.com/home

Google Inc. (2004) Google Books. Retrieved from the World Wide Web October 31, 2013: http://books.google.com/

Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M., & Valduriez, P. (2010) Streamcloud: A large scale data streaming system. *30th International Conference on Distributed Computing Systems*, pp 126-137.

Ho, R. (2010) BigTable Model with Cassandra and HBase. *DZone*. Retrieved from the World Wide Web September 9, 2013: http://java.dzone.com/news/bigtable-model-cassandra-and

IBM (2012) *What is Big Data ? Bringing Big Data to the Enterprise*. Retrieved from the World Wide Web October 31, 2014: http://www-01.ibm.com/software/data/bigdata/

Jin, A., Cheng, C., Ren, F., & Song, S. (2011) An index model of global subdivision in cloud computing environment. *19th International Conference on Geoinformatics*, pp 1-5.

Lakshman, A. & Malik, P. (2011) *The Apache Cassandra Project*. Retrieved from the World Wide Web October 31, 2014: http://cassandra.apache.org/

OSGi Alliance (2010). Semantic Versioning. *Technical Whitepaper, Revision 1.0*. Retrieved from the World Wide Web October 31, 2014: http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf

Ramesh, S. (2011)CrowdDB – Answering Queries with Crowdsourcing. Master's Thesis. Retrieved May 4, 2014: http://www.kutter-fonds.ethz.ch/App_Themes/default/datalinks/RameshMT2011.pdf

Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., & Christos, K. (2007) Evaluating MapReduce for Multi-core and Multiprocessor Systems. *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture,* pp 13-24. doi>10.1109/HPCA.2007.346181.

Robert, H. (2012) It's time for a new definition of Big Data. *MIKE2.0: The open source standard for Information Management*.Retrieved from the World Wide Web October 31, 2014: http://mike2.openmethodology.org/

Shaughnessy, S. T. (2012) Database system providing high performance database versioning. U.S. Patent No. 8,117,174. Washington, DC: U.S. Patent and Trademark Office.

solidIT. (2014) DB-ENGINES: *Knowledge Base of Relational and NoSQL Database Management Systems*. Retrieved from the World Wide Web September 3, 2014: http://db-engines.com/en/ home

Stanev, T. I. (2012) Database versioning system. U.S. Patent Application 13/354,885.

Suryavanshi, R., &Yadav, D. (2012) Modeling of Multiversion Concurrency Control System Using Event-B. *In Proceedings of the Federated Conference on Computer Science and Information Systems*, Warsaw, Poland, pp 1397–1401.

Villars, R. L., Olofson, C. W., & Eastwood, M. (2011) Big Data: What it is and why you should care. *IDC White Paper*. Framingham, MA: IDC.

Watters, A. (2010) The Age of Exabytes: Tools and Approaches for Managing Big Data (Website/Slideshare). *Hewlett-Packard Development Company*.

Weiss, R., Zgorski, L. J. (2012. *Obama Administration Unveils "BIG DATA" Initiative: Announces $200 Million In New R&D Investments*. Retrieved from the World Wide Web October 31, 2014: http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release.pdf

Wetherell, C. (2005)Google Reader:Two weeks.Reader is using Google's BigTable in order to create a haven for what is likely to be a massive trove of items. Retreived from the World Wide Web September 4, 2013: http://googlereader.blogspot.com/2005/10/google-reader-two-weeks.html

Whitchcock, A. (2005) Google's BigTable. *There are currently around 100 cells for services such as Print, Search History, Maps, and Orkut.*

White, T. (2012) Hadoop: The Definitive Guide. *O'Reilly Media*, p 3. ISBN 978-1-4493-3877-0.

(Article history: Received 7 August 2014, Accepted 12 October 2014, Available online 24 November 2014)