Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Review:

Big data storage technologies: a survey

Aisha SIDDIQA[†‡1], Ahmad KARIM[2], Abdullah GANI[1]

([1]*Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia*)

([2]*Department of Information Technology, Bahauddin Zakariya University, Multan 60000, Pakistan*)

[†]E-mail: aasiddiqa@gmail.com

Received Dec. 8, 2015; Revision accepted Mar. 28, 2016; Crosschecked Aug. 8, 2017

Abstract: There is a great thrust in industry toward the development of more feasible and viable tools for storing fast-growing volume, velocity, and diversity of data, termed 'big data'. The structural shift of the storage mechanism from traditional data management systems to NoSQL technology is due to the intention of fulfilling big data storage requirements. However, the available big data storage technologies are inefficient to provide consistent, scalable, and available solutions for continuously growing heterogeneous data. Storage is the preliminary process of big data analytics for real-world applications such as scientific experiments, healthcare, social networks, and e-business. So far, Amazon, Google, and Apache are some of the industry standards in providing big data storage solutions, yet the literature does not report an in-depth survey of storage technologies available for big data, investigating the performance and magnitude gains of these technologies. The primary objective of this paper is to conduct a comprehensive investigation of state-of-the-art storage technologies available for big data. A well-defined taxonomy of big data storage technologies is presented to assist data analysts and researchers in understanding and selecting a storage mechanism that better fits their needs. To evaluate the performance of different storage architectures, we compare and analyze the existing approaches using Brewer's CAP theorem. The significance and applications of storage technologies and support to other categories are discussed. Several future research challenges are highlighted with the intention to expedite the deployment of a reliable and scalable storage system.

Key words: Big data; Big data storage; NoSQL databases; Distributed databases; CAP theorem; Scalability; Consistency-partition resilience; Availability-partition resilience

http://dx.doi.org/10.1631/FITEE.1500441 **CLC number:** TP311.13

1 Introduction

Nowadays, big data is the frontier topic for researchers, as it refers to rapidly increasing amounts of data gathered from heterogeneous devices (Chen and Zhang, 2014). Sensor networks, scientific experiments, websites, and many other applications produce data in various formats (Abouzeid *et al.*, 2009). The tendency to shift from structured to unstructured data (Subramaniyaswamy *et al.*, 2015) makes traditional relational databases unsuitable for storage. This inadequacy of relational databases motivates the development of efficient distributed storage mechanisms. Provision of highly scalable, reliable, and efficient storage for dynamically growing data is the main objective in deploying a tool for big data storage (Oliveira *et al.*, 2012). Thus, innovative development of storage systems with improved access performance and fault tolerance is required.

Big data has influenced research, management, and business perspectives and has captured the attention of data solution providers toward the deployment of satisfactory technologies for big data storage (Sakr *et al.*, 2011). Relational databases have been very efficient for intensive amounts of data in terms of storage and retrieval processes for many decades (Vicknair *et al.*, 2010). However, with the advent and accessibility of the Internet, technology to the public has turned the structure of data towards schema-less,

‡ Corresponding author
ORCID: Aisha SIDDIQA, http://orcid.org/0000-0002-1016-758X
© Zhejiang University and Springer-Verlag GmbH Germany 2017

interconnected, and rapidly growing. Apart from that, the complexity of data generated by web resources does not allow the use of relational database technologies for analyzing image data (Xiao and Liu, 2011). The exponential growth, lack of structure, and the variety in types bring data storage and analysis challenges for traditional data management systems (Deka, 2014). Transformation of big data structures to relational data models, strictly defined relational schemas, and complexity of procedures for simple tasks are the rigid features of relational databases (Hecht and Jablonski, 2011), which are not acceptable to big data.

NoSQL technologies introduce flexible data models, horizontal scalability, and schema-less data models (Gorton and Klein, 2015). These databases aim to provide ease in scalability and management of large-volume data (Padhye and Tripathi, 2015). NoSQL databases offer a certain level of transaction handling so that they are adequate for social networking, e-mail, and other web-based applications. To improve the accessibility of data to its users, data are distributed and replicated in more than one site. Replication on the same site not only supports data recovery in case of any damage but also contributes in high availability if replicas are created on different geographic locations (Tanenbaum and van Steen, 2007; Turk *et al.*, 2014). Consistency is another aspect of distributed storage systems when data has multiple copies and keeping the data up-to-date on each site becomes more challenging. Brewer (2012) pointed out that preference to either availability or consistency is a common design objective for distributed databases whereas network partitions are rare.

To date, NoSQL technologies have been widely deployed and reported as surveys in the literature, yet state of the art does not provide an in-depth investigation into the features and performance of NoSQL technologies. For instance, Sakr *et al.* (2011) presented a survey highlighting features and challenges of a few NoSQL databases to deploy on the cloud. Deka (2014) surveyed 15 cloud-based NoSQL databases to analyze read/write optimization, durability, and reliability. Han *et al.* (2011) described seven NoSQL databases under key-value, column-oriented, and document categories at an abstract level and classified them with the CAP theorem. Similarly, Chen *et al*. (2014) surveyed nine databases under the same three categories as described by Han *et al.* (2011) in the storage section of their survey. Another significant contribution in reviewing big data storage systems was made by Chen *et al*. (2014), who explained the issues related to massive storage, distributed storage, and big data storage. Their review also covers some well-known database technologies under key-value, column-oriented, and graph data models and categorizes them with Brewer's CAP theorem. However, these three studies did not cover a large number of NoSQL databases in key-value, column-oriented, and document categories. Moreover, graph databases are not considered in these studies. In contrast, Vicknair *et al.* (2010) and Batra and Tyagi (2012) have studied Neo4j, which is a graph database, in comparison with the relational database to observe full-text character searches, security, data scalability, and other data provenance operations. Zhang and Xu (2013) highlighted the challenges (i.e., volume, variety, velocity, value, and complexity) related to storage of big data over distributed file systems in a different perspective. However, this survey did not aim to report the performance of existing NoSQL databases for the described challenges. Many other comparative studies are present in the literature, analyzing the performance of some specific category or limited features of NoSQL databases. However, the state of the art does not report any detailed investigation of a vast set of performance metrics covering a large number of storage technologies for big data.

Therefore, in this paper we highlight the features of distributed database technologies available for big data and present a more comprehensive review. We thoroughly study 26 NoSQL databases in this survey and investigate their performance. Moreover, we describe a number of recent widely spread storage technologies for big data under each data model such as key-value, column-oriented, document, and graph along with their licensing. In addition to that, we have strengthened our analysis with a discussion on the existing and recent trends in Brewer's theorem. In accordance with Brewer's recent explanation for distributed system characterization, we have highlighted each NoSQL database as either consistent or highly available. Therefore, the remarkable contribution of our work is to provide awareness for big data analysts to choose a storage option from a vast variety of databases with better tradeoff between consistency

and availability. Furthermore, this study helps researchers understand and leverage an optimum storage solution for their future research work.

Following are the key objectives of this paper: (1) to investigate storage structures of a wide range of technologies in a big data environment; (2) to highlight distinctive properties of each storage technology; (3) to develop the taxonomy and evaluate big data storage technologies according to the well-known Brewer theorem presented for distributed systems; (4) to identify the challenges and research directions for coping with big data storage in the future.

The rest of the paper is organized as follows: Section 2 describes the evolution of big data storage technologies and their distinctive features over relational databases. Contemporary storage technologies for big data are also detailed in Section 2. Section 3 presents the taxonomy and categorization based on adopted data model and licensing. Section 4 describes Brewer's CAP theorem for distributed systems along with its new explanation. Storage technologies are investigated and analyzed to suggest a type based on Brewer's categorization. Section 5 summarizes the discussion and highlights future research challenges. Section 6 concludes the discussion.

2 Evolution of big data storage technologies

In this section we discuss the technological shift from relational, well-structured databases to non-relational, schema-less storage technologies. The drive and challenges due to big data are summarized. Moreover, the prominent features of storage technologies specified for big data are highlighted. Most commonly used big data storage technologies are also elaborated upon.

Over past few decades, relational databases have been used as well-structured data management technologies. They have been recommended for performing data management operations on structured data (Deagustini *et al.*, 2013). Datasets such as the Internet Movie Database (IMDB, 2015) and MovieLens (MovieLens, 2015) are available for being manipulated using relational databases. Big data and emerging technologies like cloud computing allow data to be captured from interactive and portable devices in various formats (Kaisler *et al.*, 2013; Chen and Zhang, 2014). The data come with the new challenges of fast retrieval, real-time processing, and interpretation over a large volume (Kumar, 2014). Unfortunately, relational databases could not evolve as fast as big data. Moreover, the support to fault tolerance and complex data structures is not satisfactory for heterogeneous data (Skoulis *et al.*, 2015). Furthermore, the schema of relational databases does not support frequent changes (Neo4j, 2015). Google, Amazon, and Facebook are some of the well-known web data repositories. Their processing and dynamic scalability requirements are beyond the capabilities of relational databases (Pokorny, 2013). Thus, continuously growing data that come with heterogeneous structures need a better solution. We summarize the comparison between relational databases and big data storage systems in Table 1 using a SWOT analysis.

Big data is essential for enterprises to predict valuable business outcomes. To meet the challenges of big data, NoSQL (not only SQL) databases have emerged as enterprise solutions. NoSQL databases overcome the problems of relational databases and offer horizontally scalable, flexible, highly available, accessible, and relatively inexpensive storage solutions (MacFadden, 2013). Thus, NoSQL databases have become the mostly adopted technologies for storing big data. Unlike relational databases, these technologies offer support to a large number of users interacting with big data simultaneously. NoSQL databases are great in achieving consistency, fault tolerance, availability, and support to query (Cattell, 2010). They also guarantee some distinctive features over relational databases: scalability, availability, fault tolerance, consistency, and secondary indexing.

2.1 Distinctions of big data storage technologies

It is very common to consider scalability, reliability, and availability as the design goals for a big data storage technology. However, it is also observed that consistency and availability influence each other in a distributed system, and one of them is compromised (Diack *et al.*, 2013). According to the nature of big data, a single server is not a wise decision for storage, and it is better to configure a cluster of multiple hardware elements as the distributed storage system. To discuss storage technologies for big data, the description of features for distributed NoSQL systems provided in the literature is also significant.

Table 1 SWOT analysis of relational databases and big data storage systems

	Traditional database systems	Big data storage systems
Strengths	Support highly structured data stored and processed over an auxiliary server Vertical scalability with extendible processing on a server Specialized data manipulation languages Specialized schema	Support heterogeneous structured data Horizontal scalability with extendible commodity servers Support data-intensive applications Simultaneous accessibility Reliability and high availability High fault tolerance Eventual consistency
Weaknesses	Performance bottleneck Processing delays Increased deadlocks with growth of data Limited storage and processing capacity Co-relations which hinder scalability Expensive join operations for multidimensional data	No compliance with ACID due to scalability and performance
Opportunities	Support complex queries Atomicity in complex transactions Built-in deployment support	Improved query response times Simplicity in storage structures Data-intensive
Threats	Extensive volume of data for storage with dynamic growth Frequently changing schema Complex data structures More concurrent access needs Frequent I/O needs Real-time processing needs Consistency of a large number of storage servers	Large number of small files Deployment may need community support

For this reason, we explain these features in distinction. Out of them, consistency, availability, and partition resilience are further considered to examine the applicability of Brewer's theorem for current big data storage technologies in Section 4.

Scalability refers to support to growing volumes of data in such a manner that a significant increase or optimization in storage resources is possible (Putnik *et al.*, 2013). The paradigm shift from batch processing to streaming data processing shows that the data volume is continuously increasing. Referring to our previous work (Gani *et al*., 2015), the volume for future big data will be at zettabyte scale and the storage requirements will increase with such volume. As far as the availability of a system is concerned, it suggests quick access to data storage resources (Bohlouli *et al.*, 2013). For this purpose, data are replicated among different servers, which may be placed on the same location or distant locations to make the data highly available to users at their nearby sites, thus increasing big data retrieval efficiency (Azeem and Khan, 2012; Wang *et al*., 2015). In other words, minimum downtime and promptness of a system for ad hoc access requests define its availability (Oracle, 2015a).

Node failure is very common in distributed storage systems. To make it fault-tolerant, multiple copies of data are created and placed on the same node and/or different nodes of the storage cluster. Replication not only makes the system highly available but it is also useful for fault tolerance (Hilker, 2012). Furthermore, NoSQL databases offer very flexible schema and data relationships and are not as complex as relational databases (Kumar, 2014). Nowadays, data not only comprise tuples; documents and objects are also part of big data. Therefore, a predefined schema cannot deal with varying data structures (Cattell, 2010).

Regardless of the distribution, storage systems for big data ensure the data be complete and correct. Changes made by users are committed under defined rules (Oracle, 2015a). Eventual consistency has become a widely adopted mechanism to implement consistency in NoSQL databases. Changes are propagated eventually, and the system becomes consistent after propagating the changes. However, instantane-

ous propagation leads to the development of a strongly consistent system, yet it results in frequent access locks. In addition to fault tolerance and consistency, indexing is worthwhile for big data storage. Indexing is a method that improves the performance of data retrieval. In relational databases, primary keys are sufficient to perform search operations (Oracle Secondary, 2015). However, with the advent of big data, which comes with new challenges of heterogeneity in data structures, primary key indexing is not the solution. Secondary indexes are mostly created automatically by the system, and keys other than the primary key are produced.

2.2 Contemporary big data storage technologies

Research outcomes of exploring storage technologies for big data advocate different aspects of designing storage mechanisms. These reliable and highly available mechanisms contribute to improving data access performance. Improved data access performance drives better quality of data analysis. These technologies offer scalable storage solutions for growing big data with enhanced data structures and support fault tolerance. This section provides a brief explanation of storage systems for big data in each category. In accomplishing the design goals, current storage technologies are described to review their feasibility for big data. Their storage structure and outstanding features to support scalable resource provisioning for big data are also described here.

The Google File System (GFS) is a proprietary system developed by Google Inc. (Ghemawat *et al.*, 2003) to manage data-intensive applications in a distributed manner. It is designed to satisfy the storage needs for steadily growing data as a significant objective along with other features provided by contemporary techniques. Current and future estimated workloads are analyzed to develop such a distributed file system. To deal with the commodity component failure problem, GFS facilitates continuous monitoring and ensures detecting errors, tolerates component faults, and recovers them automatically. GFS adopts a clustered approach that divides data chunks into 64-KB blocks and stores a 32-bit checksum for each block. As shown in Fig. 1, these checksums are stored on servers as part of metadata to ensure integrity. Moreover, the chunks are replicated to avoid chunk server faults and for availability and reliability (Dean and Ghemawat, 2008). Such systems are supposed to handle large-volume data where many kilo-byte size files become a challenge for them. However, GFS guarantees support to manage these small files along with appending new data concurrently for large files even when they are read/write-intensive.

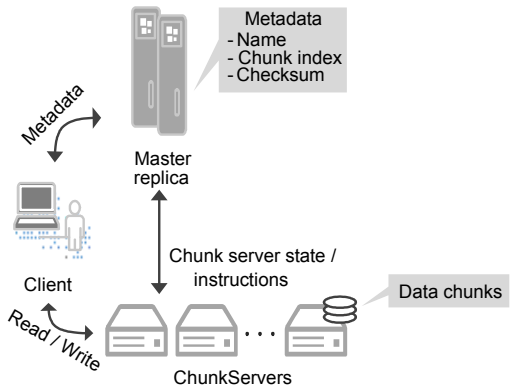

Fig. 1 Google File System (GFS) architecture (Ghemawat *et al.*, 2003)

The Hadoop Distributed File System (HDFS) is developed as an inspiration of GFS. HDFS is a distributed, scalable storage system designed as a core of Apache Hadoop to run on inexpensive commodity hardware, initially designed as infrastructure of Apache Nutch. HDFS is a suitable solution for data-intensive applications, typically at gigabyte to terabyte scales, which require high throughput. HDFS provides quick fault detection and automatic recovery, as it comprises a large number of components. However, there is a probability of block failure and nonfunctioning (Borthakur, 2008). Block replication is offered to avoid node failure and unavailability or loss of data (Shvachko, 2010). Replication ensures not only the availability but also the reliability of the system and it is automatically handled by HDFS NameNode. Rather than just being a storage layer of Hadoop, HDFS is a standalone distributed file system that helps improve the throughput of the system. HDFS has a namespace-separated architecture. Metadata is stored on the master node, which is called NameNode, whereas block-split files are stored on a number of DataNodes. NameNode performs mapping of data on DataNodes and namespace operations such as open, close, and rename file. DataNodes fulfill read–write requests and create block and replicas. The architecture of HDFS is shown in Fig. 2.

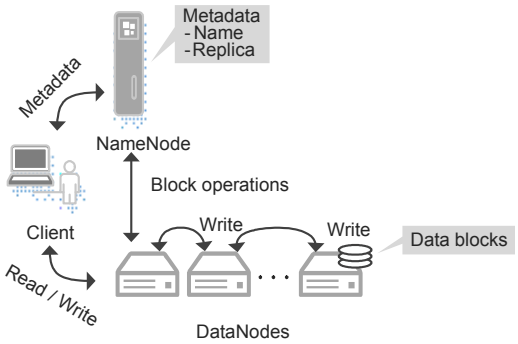

Fig. 2 Hadoop Distributed File System (HDFS) architecture (Borthakur, 2008)

BigTable is another product developed by Google Inc. (Chang *et al.*, 2008) to provide a flexible and high-performance storage for large-scale structured data spread over a large number of commodity servers. BigTable is a highly adaptable, reliable, and applicable storage system to manage petabyte scale data on thousands of machines. BigTable has proven to be a high-performance and available data storage. Therefore, many Google applications and projects that might have throughput or latency requirements are using BigTable with different workloads. Moreover, BigTable provides dynamic control over data placement, representation, indexing, and clustering.

HBase is a column-oriented data store developed by Apache (Apache Software Foundation, 2015). HBase implements a BigTable storage system and is designed to handle storage needs of big data in the Apache project (George, 2011). HBase provides a scalable, distributed, fault-tolerant, and random read–write oriented access to big data on top of Hadoop and HDFS (Fig. 3) (Taylor, 2010). HBase uses the underlying HDFS from Hadoop to store table data. Its master component assigns regions to RegionServers and keeps track of these servers. RegionServers receive region information and manage read/write requests from clients on different regions (Khetrapal and Ganesh, 2006). As far as concurrency is concerned, HBase has a remarkable support to read-intensive transactions (Ruflin *et al.*, 2011). HBase is more concerned about distribution and uses a clustered approach for data management; that is why it is a potential solution for a remarkably large number of rows of data.

Hypertable also implements BigTable (Hypertable, 2015). Hypertable is a distributed database that provides a very good support to consistency of stored data (Han *et al.*, 2011). Hypertable is designed to run and is compatible with many distributed file systems such as GFS, HDFS, and CloudStore. Hypertable stores data in the form of tables and splits the tables to achieve distribution and scalability. Fig. 3 describes the relationship between the components of the Hypertable system. HyperSpace works as a lock manager and ensures high availability and consistency through distributed configuration of replicas by using a distributed consensus protocol. There is no effect on client data transfer if the master becomes unresponsive for a short period. This is preferable to have more Masters to achieve high availability requirements (Khetrapal and Ganesh, 2006).

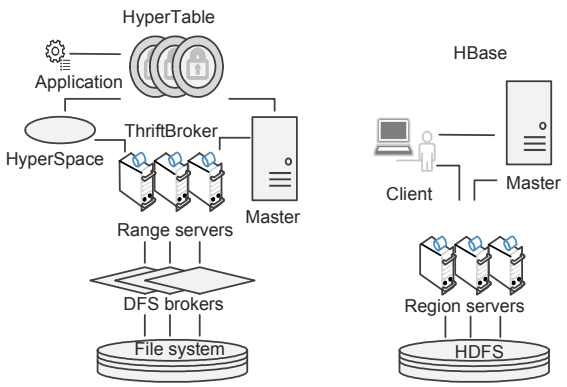

Fig. 3 Two implementations of BigTable (George, 2011; MongoDB, 2015)

MongoDB is an open-source NoSQL database designed by Mongo, Inc. as a highly available, scalable, and fault-tolerant document-oriented solution (MongoDB, 2015). MongoDB derives the characteristics of MySQl with the JSON data model (Ruflin *et al.*, 2011). Therefore, MongoDB has the same horizontal scalability, ease of agile development with dynamic schema support to all kinds of document data, and efficiency in manageability as MySQL offers. Furthermore, indexing, dynamic and ad hoc queries, aggregation, and dynamic updates are some of the vast MySQL facilities that are adopted by MongoDB with slight modification. MongoDB stores documents as data in binary representation called BSON, which makes it easy to map data from applications to databases (Kristina and Michael, 2010). Moreover, robustness and availability of data are achieved with a replica set, and multiple replicas are available when the primary server is not responding. This database provides support for storing and

combining multiple structure data, whereas the indexing is still feasible (Banker, 2011).

Terrastore is another document-based open-source distributed storage system, developed by Terracotta, Inc. (Terrastore, 2015). Terrastore offers support with high scalability as well as consistency and dynamic clustering during execution. Terrastore is built on top of the in-memory storage, Terracotta. To provide scalability, Terrastore offers not only the deployment of a single cluster but also support to multiple clusters. Furthermore, load balancing for data storage over nodes is automatically performed when nodes connect or disconnect from the clusters (Bossa, 2009). Terrastore is optimized for consistency, as it performs consistent hashing for partitioning and does not create replicas. However, data are redistributed when nodes join or leave the cluster. Like DynamoDB, Voldemort, and Cassandra, the Terrastore node nearby a client forwards the client requests to the serving node and it is not necessary for the client to have this routing knowledge. Although the built-in support to data distribution and partitioning makes Terrastore better than Memcached, Terrastore has more distribution overhead than Redis, which lowers its performance (Alex and Ana-Maria, 2009). Consequently, Terrastore is not considered a good approach for large amount of document data (Bossa, 2009).

HyperGraphDB is an implementation of the hypergraph model to develop an open-source graph database for artificial intelligence and web semantic projects (Iordanov, 2010). This graph database uses Berkeley DB for storage and not only provides storage for arbitrary data but also supports data mapping between host language and storage. In addition, an essential customizable indexing feature is provided by HyperGraphDB so that efficient data retrieval and graph traversal are achieved. HyperGraphDB uses a key-value mechanism to store graph information such as nodes and edges as a key (Dominguez-Sal *et al.*, 2010). In contrast to HBase, HyperTable, Redis, and other master–slave storage systems and access architectures, HyperGraphDB implements a peer-to-peer data distribution mechanism. Each peer works independently, and updates are performed asynchronously and eventually (HyperGraphDB, 2010).

InfiniteGraph is a distributed object database designed for graph data that offers high scalability as well as complex search and traversing efficiency (InfiniteGraph, 2014). InfiniteGraph is a graph database where data and computation load are distributed among storage locations and runtime configurable zones. To achieve scalability, InfiniteGraph implements a model-based technique. Moreover, an InfiniteGraph database facilitates the creation of a custom model for frequently accessed elements so that query performance is improved. These frequently accessed elements can further be separated from other elements to handle locks. Along with custom models, the indexing framework of InfiniteGraph also contributes to better query performance (Objectivity, Inc., 2012). InfiniteGraph database users know that their growing graph data are stored with some schema and they can easily perform normalization and other presentation operations on their data (Fulton, 2011). Furthermore, because of deploying its model on cloud, development and testing of databases over InfiniteGraph is without any charge.

Rocket U2 is a document-oriented, proprietary database management suite that offers two associated platforms of multivalue databases: UniData and UniVerse (RocketSoftware, 2015). Rocket U2 ensures availability of the system after any disconnection, which makes it reliable and robust to immediately recover from any downtime (RocketSoftware, 2014a). According to a case study (RocketSoftware, 2014b), Rocket U2 provides a scalable and flexible system where changes can be made anytime according to the requirements. This flexibility allows easy updating in applications, and the database does not need to be rebuilt when changes occur. Multivalue products are integrated with many pieces of codes, which reduces the developers' effort of coding. Furthermore, Rocket U2 databases sustain their performance in terms of response time even when the size of data grows. However, support to data partitioning is not available with Rocket U2.

Scalaris is a scalable, distributed, and highly available key-value store that is designed to fulfill the intensive read/write requirements of enterprises (Schütt *et al.*, 2008). For concurrent access and to support intensive transaction, it is common to face node failures. However, Scalaris manages to obtain consistency for critical write operations by implementing a distributed transaction protocol and a structured overlay network on Erlang. The structured

overlay network is implemented as the first layer of Scalaris architecture and provides support to a range of queries. The architecture of Scalaris also applies replication to make the system highly available and fault-tolerant. Regardless of the challenges of write-intensive transactions, the Paxos consensus protocol helps ensure consistency with reduced overhead (Scalaris, 2015).

To achieve high performance, custom data management, and scalability for applications, Berkeley DB provides an embedded support to key-value databases via libraries. Although Berkeley DB offers an open-source license, this license is constrained to the development of open-source applications. For third-party distribution, Berkeley DB offers commercial licensing facility. Berkeley DB was initially licensed under Sleepycat Software but later owned by Oracle. The products developed using Berkeley DB have simple data access and management procedures. Furthermore, due to customized configuration with no overhead, these products fulfill varying development needs such as local or distributed storage and storage capacity configuration, and many other options are available when Berkeley DB is used (Oracle, 2015b). As far as other features related to data distribution are concerned, Berkeley DB ensures high throughput for data access rate, non-blocking consistency, reliability, fault tolerance, and availability. Like relational databases, Berkeley DB has a good support to data access efficiency, transactions, and recovery mechanism (Seltzer and Bostic, 2015). However, provision to these relational databases-like features is packaged in libraries, and developers are given the configuration choice according to their requirements. Regardless of the features provided by Berkeley DB, it is noticeable that Berkeley DB does not support synchronization with other databases. Meanwhile, data partitioning is not facilitated by Berkeley DB (Oracle, 2015c).

DynamoDB is a widely used, fully distributed, schema-free NoSQL database tool by Amazon (Sivasubramanian, 2012). DynamoDB is predominantly used for storing unstructured, variable, and scalable data (Baron and Kotecha, 2013). DynamoDB offers infinite scaling capacity for data storage and access rate. It is applicable when frequent updates on distributed sites, efficient indexing, and adaptive scalability are required. Availability and durability can be achieved through automatic replication over multiple zones in a region (Niranjanamurthy *et al.*, 2014). Regardless of the request size, DynamoDB ensures stable performance and visible resource utilization. Administrative tasks, such as hardware provisioning, setup and configuration, replication, software patching, and scaling, can be offloaded to Amazon Web Service (AWS). DynamoDB is an efficiently managed database; that is why it is suitable for scalable applications with exponentially growing data (Vyas and Kuppusamy, 2014).

Qualcomm has developed a document-oriented database named Qizx for large-volume, high-performance, and transaction-intensive XML data (Qualcomm, 2014a). Qizx is well suited for enterprises with fast data access and retrieval requirements. For this purpose, Qizx has automatic full-text and customizable indexing capability, which makes it efficient in query responses. In addition, support to terabyte-scale data and other features related to distribution are taken into consideration while designing this database. Qizx's 'shared-nothing' clustering architecture allows each node or host in cluster to fully replicate the database to achieve high availability (Qualcomm, 2014b). However, these share-nothing hosts serve client read requests, and one dedicated server is responsible for update operations. Consequently, a consistent, fault-tolerant, and up-to-date replicating database system is achieved.

An industry-leading, broadly used, schema-less graph database that is effectively replacing relational databases is Neo4j (Neo4j, 2015). By analyzing complex data relationships with Neo4j, enterprises gain value benefits. In regard to scalability (Montag, 2013), high transaction load, concurrency, and performance for read request workloads, Neo4j not only competes other graph databases but also affects improvements on its older versions (Zicari, 2015). Support to write-intensive transactions without blocking is achieved with the help of buffering. In spite of its benefits, Neo4j is time consuming in setting up a ready-for-production and reliable database.

For real-time web applications, RethinkDB is the first open-source document database that supports complex queries in an efficient way (RethinkDB, 2015). RethinkDB reverses traditional storage systems and implements append-only nested data storage structure, which makes it more consistent, robust, and

easily replicable (Walsh *et al.*, 2009). RethinkDB offers easy scalability and faster response to client's real-time queries. Although RethinkDB also uses the JSON data model, query operation processing is slower than that of MongoDB. However, RethinkDB has a real-time architecture and, due to its append-only storage structure, RethinkDB incurs less scaling cost as compared to MongoDB. Furthermore, advanced query language and administration are better than with MongoDB. Despite the benefits obtained by RethinkDB, RethinkDB does not perform very well for compute- and write-intensive applications.

Aerospike is the first flash-optimized, open-source, key-value data store for real-time data, which offers scalability and reliability with very low cost (Aerospike, 2015). Aerospike has a 'shared-nothing' architecture, which supports petabyte scale volume of data with reliability and linear scalability (Aerospike, 2012). Similar to Cassandra, a column-oriented database, Aerospike is designed to provide scalability, replication, automatic node partition recovery, and high availability. However, Aerospike is better than Cassandra in terms of read and write workload, consistency with jitters, and cost for read and write operations.

The first multi-model, open-source, and highly scalable database for document data with an extended, transparently managed graph layer to provide connections between records as relationships is OrientDB (OrientDB, 2015). The embedding graph layer makes OrientDB not only persistent and compact but also fast in data traversal and managing data relationships without increasing the cost. The multi-model platform of OrientDB makes it categorizable as a document-oriented graph database. Regardless of volume of data, OrientDB has a distinct speed in data storage as well as read and write transactions. To ensure performance and scalability, clustering and replication over heterogeneous servers is a significant benefit achieved by OrientDB (SD Times Newswire, 2013). OrientDB has many features similar to Couchbase document store, which is derived from CouchDB. Both have a schema-free model, support to secondary indexes, sharding for data partitioning, and the master–master replication model. However, OrientDB has additional features such as a multi-model approach to support all operating systems, which are not provided by Couchbase.

AllegroGraph is a graph database efficiently designed for RDF semantic web applications. AllegroGraph ensures persistency, memory utilization performance, and high scalability over millions of quads (AllegroGraph, 2015). Furthermore, automatic compute and storage resources management in AllegroGraph are the key performance fundamentals. AllegroGraph offers not only an enterprise and commercial version but also an open-source product. However, the open-source version of AllegroGraph is limited in scalability. Hybrid data representation, which covers both relational model and triple store, optimizes the range query performance in AllegroGraph (Aasman, 2008).

Redis is an alternative open-source cache and key-value store for big data, which provides an efficient data structure for indexing to speed up query operations and response (Carlson, 2013). Somehow, the functionality of Redis is similar to that of Scalaris. Redis has considerable support to replication in a master–slave environment. However, providing support to multiple data structure makes it a predominant choice for frequent data access situations. Linux is the recommended operating system for its deployment, as Redis is developed and tested on the Linux platform (Excoffier and Lischer, 2010). In the case of more scalability requirements when systems become out of RAM, Aerospike in-memory store is preferable to Redis. Redis is much closer to MemcacheDB in fast data access, as the whole data resides in the memory. However, Redis has some other powerful features such as built-in persistency and support to more datatypes. Unlike MemcacheDB, Redis is persistent as a real database and data will not be lost with restart. Furthermore, support to more data types is a unique property of Redis. Thus, Redis becomes a best choice when high scalability, heterogeneity in platform, servers and application, and in-memory data are required.

Voldemort is a general-purpose, distributed data storage solution for large-scale data (Sumbaly *et al.*, 2012; Voldemort, 2015). The routing module of Voldemort is responsible for performing data clustering and replication and follows the same procedure as DynamoDB. Voldemort provides eventual consistency, as read/write operations can be performed on any node and for a very short period as happened when the view of data is inconsistent (Bunch *et al.*,

2010). Like MemcacheDB, Voldemort also achieves data persistence using Berkeley DB, and the replication factor for distributed hash table entries can be set by the developer.

KAI is a distributed, key-value-based data store that is developed as an implementation of Amazon's Dynamo. KAI provides high scalability and availability to websites with support to a variety of data. The name KAI is derived to elaborate its good organization of clusters where data are stored and processed in parallel in a distributed, replicated manner (SourceForge, 2015). Furthermore, load balancing, configurable scalability of nodes, fast configuration, and reliability through replication are the basic performance gains of KAI. Although KAI offers eventual consistency, the implementation of the quorum protocol helps achieve strong consistency.

Cassandra is a decentralized and highly available key-value storage system for structured data that comprises a large number of underlying data centers (Hewitt, 2010; Lakshman and Malik, 2010). It serves to provide scalability, instance storage, and improved performance for frequent read/write operation requests. Data consistency is achieved in Cassandra through periodic updates on replicating sites (Baron and Kotecha, 2013). In contrast to HBase, Cassandra provides significant support to write-intensive transactions (Ruflin *et al.*, 2011). Most significantly, Cassandra persistently deals with component failures in such large-scale systems and achieves overall reliability and scalability. Schema can be built any time, as Cassandra provides flexibility. Clustering, partitioning, and replication while taking care of fault tolerance, reduced latency, and scalability are the prominent features of Cassandra (Lakshman and Malik, 2010; Abramova and Bernardino, 2013). It is a reasonable choice for continuously growing systems such as Facebook. Facebook has deployed Cassandra, where scalability linearly depends upon the number of users and a write-intense system is needed (Armbrust *et al.*, 2009).

SimpleDB is an open-source, document-oriented database that is available as an Amazon service (Sciore, 2007). Without database administration load, SimpleDB ensures high availability and durability of data with automatic geographic replication. Furthermore, the data model is flexible, and automatic indexing is performed on the data. Thus, automatic provisioning of database administration makes application development simple via SimpleDB. Despite the ease of data management provided by SimpleDB, scalability is limited to 10 GB (Habeeb, 2010).

MemcacheDB is an open-source key-value store (Tudorica and Bucur, 2011; MemcacheDB, 2015) developed for fast and reliable storage and access of objects and data. It implements Berkeley DB to develop a general-purpose database, and many features of Berkeley DB such as transaction and replication are derived. MemcacheDB is a persistent database for dynamic, database-driven web applications (Coburn *et al.*, 2011). Memcached users can access MemcacheDB through any API, as it uses the Memcached protocol. When implementing Memcached API, a cache system for fast access from memory, it becomes easy to provide data persistency as caching is already performed by Memcached (Helmke, 2012). It applies a master–slave approach to achieve data persistency where the data read request can be fulfilled by any site but write operation is performed only at the master node. It operates with a single master and multiple replicating nodes. However, it is flexible to allow any node to become a replicating node, but it reveals a security challenge (Bunch *et al.*, 2010).

CouchDB is an open-source, distributed, scalable document database developed for performing data operations and administration on the web (Anderson *et al.*, 2010; Apache, 2015). In case of intensive access requests, CouchDB adopts a sophisticated concurrency mechanism that does not let the system fail. However, such intensive workload cause delays in overall responses. CouchDB has support to dynamic data structure, which makes it possible to define schema as required. Regardless of wide adaptability of CouchDB in web application development, it is suggested to check its feasibility for developer requirements (Wenk and Slater, 2014). Like MongoDB, CouchDB also implements the JSON data model, which helps CouchDB support semistructured data. Furthermore, CouchDB allows the storage of any kind of data as documents (Ruflin *et al.* 2011).

Riak is a key-value database that provides high availability and less cost scalability (Sheehy, 2010). Riak provides simpler data models and data conflict resolution. Riak is an advanced form of key-value databases but somehow it is categorized as a document-based database. Although Riak has more

functionalities than other key-value databases, it lacks the features of document-based databases (Cattell, 2010). Therefore, it is better to categorize Riak as a key-value database. As compared to Redis, Riak has a less complex key construct, which is a feature of document-based databases (Ruflin *et al.*, 2011). This less complexity to key construct makes the data model of Riak flexible, and changes in the data model do not affect the indexes.

From an analytics perspective, tensor-based models are widely adopted to represent arbitrary relationships in data entities and to traverse graphs (Cichocki, 2014). Big data has high dimensionality rather than two-tuple relationships and brings the challenge of analyzing such multi-relational datasets. Therefore, tensor-based models are being used for big data representation. SciDB (Cudré-Mauroux *et al.*, 2009), an implementation of the tensor model, is designed to manage multidimensional data as an array database, which creates chunks of arrays for parallel processing. SciDB supports many data manipulation operations such as linear algebra operators for chunk-based *N*-dimensional arrays. Attributes and dimensions of data are flexible and can be added or removed any time. SciDB is an append-only array database for scientific data, and the changes or corrections made in data are stored as new versions (Stonebraker *et al.*, 2013).

In this section, we have provided a comprehensive account of current storage technologies for big data. Table 2 provides a brief summary of above discussed storage technologies available for big data. It highlights the design objectives and characteristics of these technologies. It comprises four vertical divisions, providing the name of the storage technology, the vendors, and the design goals. Development of distributed storage mechanism, which partitions the data in a NoSQL database format, provides scalability and efficient and fault-tolerant support to read/write operations on that data.

Table 2 provides a summary of big data storage technologies. It can be seen that most of the storage technologies are designed to ensure scalability and consistency in general. However, these storage systems support specific features such as intensive workload, high availability, data access performance, and complex query operations. We provide a comparative table identifying the application areas of each storage technology in Table 3. The following section highlights these features to categorize the above discussed storage systems. Taxonomy based on the adopted data model of big data storage technologies is presented and detailed in the next section.

3 Taxonomy of big data storage technologies

Big data storage technologies described in the previous section are categorized according to their data model and licensing in this section. There are four types of data model, key-value, column-oriented, document-oriented, and graph, whereas licensing has three categories, open source, proprietary, and commercial. We classify the storage technologies and develop taxonomy based on their data models and licensing. Additionally, we elaborate tensor decompositions and networks as representation and analysis models for multi-relational big data stored using graph-based technologies. Scalability and fast data retrieval are the significant expectations of a storage system in this regard.

However, categories of data models offer different levels of heterogeneity. The following is a description of the data models, and the example cases for each data model are presented in Fig. 4.

Key-value: Key-value databases are developed for big data storage and store not only structured but especially unstructured data in the form of a key and the corresponding value of each data record (Pokorny, 2013). These databases have suitable storage structure for continuously growing, inconsistent values of big data for which faster response of queries is required. Key-value databases provide support to large-volume data storage and concurrent query operations. Unlike block storage systems, key-value data structure stores data in small objects instead of blocks and they are easily configurable (DeCandia *et al.*, 2007). In key-value databases, values in records may differ or have different representations. In this way, this structure offers less memory consumption and the flexibility of adding more records easily. The appropriate scenario for a key-value database is when searching for more than one feature from records is needed. Most common applications of key-value storage are session information management for online games, online shopping, and other web applications where an enormous number of small-sized records are to be managed.

Table 2 Summary of big data storage technologies

Technology	Reference	Vendor	Design goals
BigTable	Chang *et al*. (2008)	Google	To bring about distribution for highly scalable, structured data
HBase	HBase	Apache	To provide consistent, random, and real-time access to scalable BigTables with read/write requests
Hypertable	Hypertable (2015)	Zvents	To provide parallel, high-performance, scalable databases for large size data; to support better querying performance for large data size
MongoDB	MongoDB (2015)	MongoDB, Inc.	To provide relational data model facilities for document-based dynamic schemas; to support quick and consistent access to data from different applications across multiple interfaces
Terrastore (in-memory)	Terrastore (2015)	Terracotta, Inc.	To achieve consistency for document data via distribution
Hyper-GraphDB	Iordanov (2010)	Kobrix Software, Inc.	To design a persistent memory model for artificial intelligence and semantic web projects; to provide both relational and object-oriented data management
InfiniteGraph	InfiniteGraph (2014)	Objectivity, Inc.	To provide persistently distributed data storage with easier traversal of complex relationships; to support complex queries over data to obtain higher values
Rocket U2	RocketSoftware (2015)	Rocket Software	To offer a scalable and stable performance for growing data volume; to provide cost-effective and highly available solutions for continuously evolving requirements
Scalaris	Sch *et al.* (2008)	Zuse Institute Berlin and onScale solutions	To achieve consistency for read/write intensive transactions; to build scalable online services
BerkeleyDB	Oracle (2015b)	Sleepycat, Oracle	To provide configurable, embedded, key-value databases with high performance; to support transparently flexible and scalable data management over applications
DynamoDB	Sivasubramanian (2012)	Amazon	To support distributed storage of scalable size of data; to improve search query performance
Qizx	Qualcomm (2014a)	Qualcomm Technologies, Inc.	To provide an enterprise-ready solution for XML data manipulation; to support text-intensive, large-volume data applications for fast data retrieval
Neo4j	Zicari (2015)	Neo Technology	To provide relational-like graph databases for intensively relating data; to support data relationship manipulation and decision making
RethinkDB	RethinkDB (2015)	RethinkDB	To support easy development of real-time web applications; to provide append only document-based storage struture
Aerospike (in-memory)	Aerospike (2015)	Aerospike, Inc.	To develop a scalable and flexible platform for web scale applications; to support reliability and consistency as traditional databases
OrientDB	OrientDB (2015)	Orient Technologies	To provide multi-model, scalable storage to achieve both graph- and document-oriented model features
AllegroGraph	AllegroGraph (2015)	Franz, Inc.	To provide a memory- and disk-efficient scalable solution for graphs
Redis (in-memory)	Carlson (2013)	Salvatore Sanfilippo	To efficiently support query operations and replication in a master–slave environment with emphasized update performance
Voldemort	Sumbaly *et al*. (2012) and Voldemort (2015)	LinkedIn	To provide distributed and consistent storage to large-scale read-only data; to support distribution transparency, failure transparency, and versioning to ensure integrity
KAI	SourceForge (2015)	Slashdot Media	To provide a highly scalable and robust solution by implementing Amazon's Dynamo
Cassandra	Hewitt (2010) and Lakshman and Malik (2010)	Apache	To provide distributed, highly available, fault-tolerant storage for big data; to improve access performance using replication and row distribution of data
SimpleDB	Sciore (2007)	Amazon	To provide automatic geographic replication for data availability and durability
MemcacheDB	MemcacheDB (2015)	Danga Interactive	To provide fast and reliable storage and retrieval
CouchDB	Apache (2015)	Apache	To provide a dynamic and self-contained schema for web documents
Riak	Sheehy (2010)	Basho Technologies	To provide high availability to applications and platforms
SciDB	Cudré-Mauroux *et al*. (2009)	Paradigm4	To support storage and manipulation of *N*-dimensional data

Table 3 Comparison and application areas of storage technologies

Technology	In-memory	On-disk	Persistence	Intensive read/write	Data partitioning	Shared nothing	Scalability	Applications
Scalaris (Sch *et al.*, 2008)	√	×	×	√	√	√	√	–Scalable online services (i.e., eBay, Amazon) –Always live databases –Frequent failure nodes
Aerospike (Aerospike, 2015)	√	×	×	√	√	√	√	–Web scale applications –SSD drives
Redis (Carlson, 2013)	√	×	√	√	√	√	×	–Session-based cache –Structured strings –LRU cache –For small data (Jing *et al.*, 2011)
Voldemort (Sumbaly *et al.*, 2012)	×	√	√	√	√	×	√	–Read-only data (Sumbaly *et al.*, 2012) –LinkedIn (Deka, 2014)
KAI (SourceForge, 2015)	×	√	√	–	√	–	×	–Web repository –Social network –Online stores
MemcacheDB	√	√	√	√	√	–	√	–Object storage –Read-only text data
Riak (Sheehy, 2010)	×	√	√	×	×	√	√	–Heterogeneous data –Always on requirements –Github, Comcast
BerkeleyDB (Oracle, 2015c)	√	√	√	×	×	×	√	–Embeddable database applications –MySQL, MemcacheDB
DynamoDB (Sivasubramanian, 2012)	×	√	√	√	√	√	√	–Unstructured variable data –Always on requirements –SSDs, e-commerce
HBase (Apache Software Foundation, 2015)	×	√	√	×	√	×	√	–Latency-tolerant applications –Versioned data –Sparse data –LinkedIn
Hypertable (Hypertable, 2015)	√	√	√	√	√	√	√	–Both structured and unstructured data –Real-time applications
Cassandra (Hewitt, 2010; Lakshman and Malik, 2010)	√	×	×	√	√	√	√	–Online interactive systems –Facebook (Armbrust *et al.*, 2009), Twitter (Deka, 2014)
BigTable (Chang *et al.*, 2008)	√	√	√	√	√	√	√	–Structured large-scale data at Google –Many Google products –Web page storage
MongoDB (MongoDB, 2015)	√	√	×	√	√	√	√	–Real-time applications –E-commerce –Google, Facebook
Terrastore (Terrastore, 2015)	√	√	√	×	√	×	√	–Structured data –No downtime requirements
RethinkDB (RethinkDB, 2015)	×	√	√	√	√	√	√	–Real-time web/mobile applications –Collaborative applications –Twitter, Github, multi-player game
SimpleDB (Sciore, 2007)	×	×	√	×	×	×	√	–Complex queries –Logs and online games
CouchDB (Apache, 2015)	×	√	√	√	√	√	√	–Versioned data –Web application –Social data

(To be continued)

Table 3

Technology	In-memory	On-disk	Persistence	Intensive read/write	Data partitioning	Shared nothing	Scalability	Applications
OrientDB (OrientDB, 2015)	√	×	×	√	√	√	√	–Embedded databases, complex data, relationships
Rocket U2 (RocketSoftware, 2015)	×	√	√	√	×	×	√	–Business information management –Mexico's emergency response systems
Qizx (Qualcomm, 2014a)	×	–	√	√	×	√	√	–For high-speed queries –Scientific data
HyperGraphDB (Iordanov, 2010)	√	√	×	–	√	–	√	–Pattern mining –Bioinformatics and semantic web
Neo4j (Zicari, 2015)	√	√	√	√	×	√	√	–Real-time recommendations –Social network –eBay, Cisco
AllegroGraph (AllegroGraph, 2015)	×	√	√	×	×	√	×	–Semantic web –Reasoning and ontology modeling –Complex data
InfiniteGraph (InfiniteGraph, 2014)	–	–	×	√	√	√	√	–Real-time search performance –Social media, location-based networking

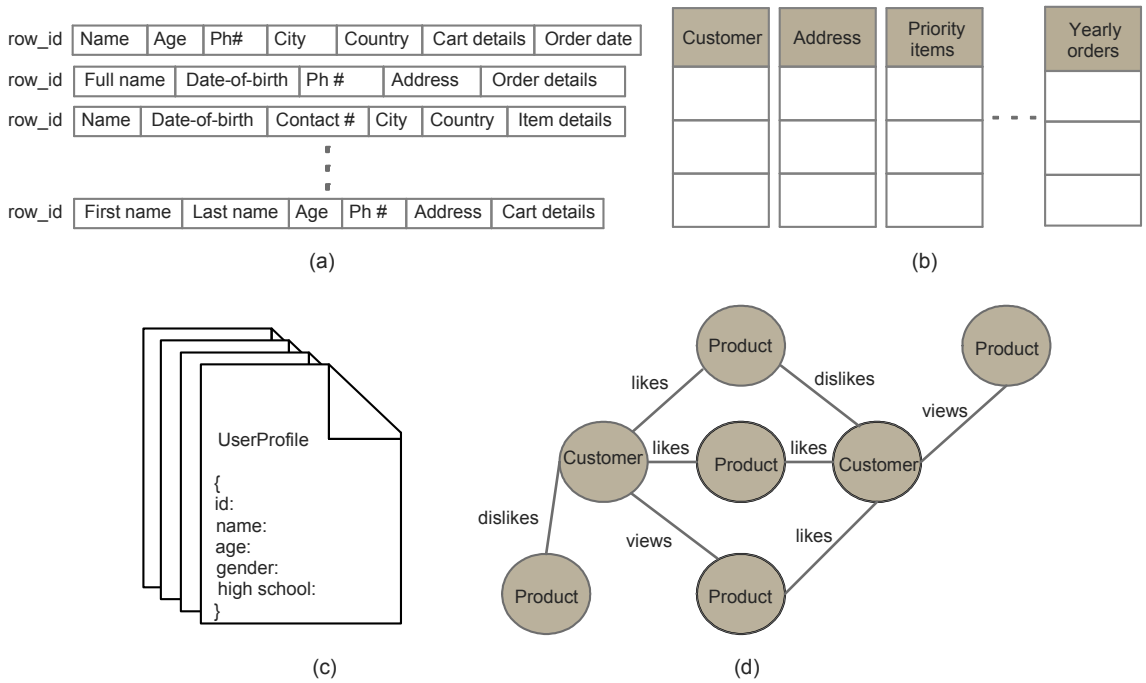

Fig. 4 Examples of data models: (a) key-value; (b) column-oriented; (c) document-oriented; (d) graph

Column-oriented: This category of NoSQL databases is suitable for vertically partitioned, contiguously stored, and compressed storage systems. Column-oriented databases store columns of data separately, unlike traditional storage where data are stored in the form of complete records. Data reads and retrieval of attributes in such systems is quite fast and comparatively less expensive, as only the relevant column is accessed and concurrent process execution is performed for each column (Abadi *et al.*, 2009; Hu and Dessloch, 2014). Column-oriented databases are highly scalable and eventually consistent. They provide support to applications for reliable and highly available storage (Dharavath and Kumar, 2015). The application areas of column-oriented databases are customer record analysis, data warehousing, patient data management, library systems, and wherever analysis is required to aggregate similar data items. Furthermore, with column-oriented structure it is convenient to add new features in all rows. For example, an online shopping website can apply aggregation on mostly viewed or ordered items in a specific time span, the trendy regions for online shopping, and the success rate of online trading in a year.

Document-oriented: Document-oriented data model is similar to key-value structure and stores data in the form of key and value as reference to the document. However, document databases support more complex queries and hierarchical relationships. This data model usually implements the JSON format and offers very flexible schema (Kristina and Michael, 2010). Although the storage architecture is schema-less for structured data, indexes are well defined in document-oriented databases. SimpleDB is the only database that does not offer explicitly defined indexes (Cattell, 2010). Document-oriented databases extract metadata to be used for further optimization and store it as documents. Use cases for document-oriented databases include user profiles on social networks, analysis of websites, and complex transactional data applications. On Facebook, as an example, users may provide less or extensive information on their profile. Each user profile will be stored as a document. The performance of heuristic schema offered by a document-oriented data storage model depends upon the user inputs and nature of queries.

Graph: Graph databases are the best choice to store data along with relationships. Graph databases offer persistent storage of objects and relationships and support simple and understandable queries with their own syntax (Zicari, 2015). Modern enterprises are expected to implement graph databases for their complex business processes and interconnected data, as this relational data structure offers easy data traversal (Neo4j, 2015). Highly frequent trading systems and recommendation systems prefer graph databases to achieve low latency. For instance, getting recommendations from customer feedback data on a commerce website requires self-joined, multilevel queries for traditional databases, which becomes a very complex operation. In contrast, for a graph database, this data manipulation is quite simple as two lines of code without affecting the structure of data.

Tensor-based models: An interesting representation of graph-oriented big data is tensor-based models, which are widely adopted models to represent arbitrary relationships in data entities and to traverse graphs (Cichocki, 2014). Dealing with high-order relationships of big data is significant in many areas of information processing such as topic modeling, social network analysis, and recommender systems. Graph-oriented systems are well suited for storing hetero-dimensional data. However, for analytical purposes, tensors are gaining attention to build open-source libraries. Lorica (2015) stated that it is useful to capture and analyze the complex relationships in data to extract more accurate information. Although there exist some efficient methods to analyze hidden variables and highly relational neural networks, the performance of tensors is beyond comparison. Scalability, accuracy, and timeliness are the reasons to prefer tensors over other methods.

Based on the specifications of each data model as described above, we present their relationship with volume and varying structure of data in Fig. 5. Key-value databases are more inclined to store each record regardless of considering completeness of data and assign a key identifier to record. However, graph databases are useful in storing well-structured, persistent objects with complex relationships. On the other hand, scalability, clustering, and data partitioning among nodes, as well as adding or removing nodes' runtime, are difficult in graph databases. Data models that support volume and scalability also have to support inconsistencies and allow storage of unstructured data over schema-less architecture.

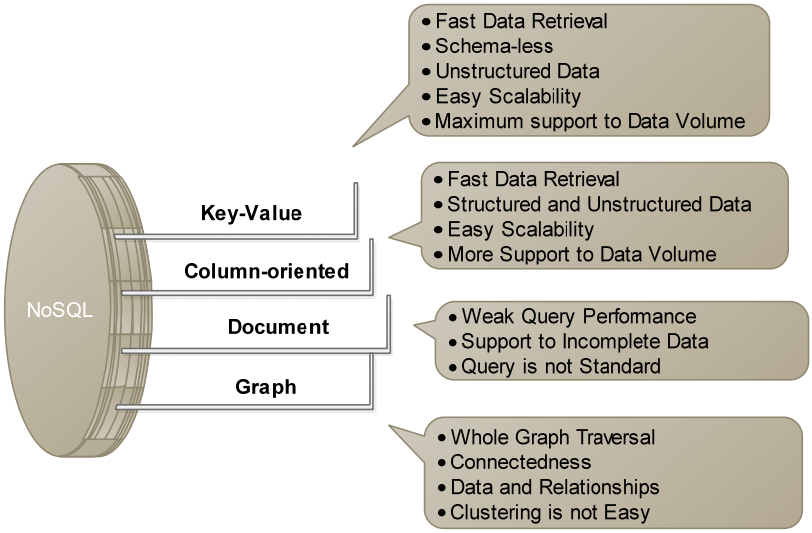

Fig. 5 Data models of NoSQL databases

In contrast, the graph data model offers support to well-structured and complex data, but the scalability is not as much as what other data models provide.

Besides data models of NoSQL databases designed as big data storage systems, we consider the type of licensing to categorize the storage mechanisms. Description of licensing is provided as the following:

Open source: An open-source system is freely available for both academia and business so that it can be used and integrated with their own piece of code or application (Milne and Witten, 2013). These systems are cost effective in terms of development and provide better quality and flexible access to even small businesses (Nagy *et al.*, 2010).

Proprietary: Unlike open-source systems, proprietary systems may be owned after paying a reasonable fee for their use. These systems are given with some legal terms and conditions to take care and only for own use and not to be modified or redistributed (Nagy *et al.*, 2010). The source codes of such systems are usually not given to the buyer.

Commercial: These systems are developed for sale. A trial part may be available free of cost but to obtain complete privilege for research and development; a user or enterprise must purchase it.

Fig. 6 elaborates the categorization of big data storage technologies that are considered and further analyzed based on Brewer's theorem in a later section. The figure lists the storage technologies available for big data and presents the data model implemented by these technologies. It also shows the classification based on types of licensing, which are open source, proprietary, and commercial. Key-value systems are developed for big data storage on NoSQL databases with different storage structures. Applications where whole records need to be accessed and analyzed are referred to as key-value storage structure. For instance, DynamoDB (Sivasubramanian, 2012) adopts key-value structure of data storage and gains better performance for storing and accessing unstructured, inconsistent data. Stable performance and visible resource utilization are ensured even when the querying workload varies. Scalaris (Schütt *et al.*, 2008), Berkeley DB (Oracle, 2015b), Aerospike (Aerospike, 2015), Redis (Carlson, 2013), Voldemort (Voldemort, 2015), KAI (SourceForge, 2015), and MemcacheDB (MemcacheDB, 2015) are also key-value storage systems as detailed in the previous section.

In column-oriented storage systems, data are vertically partitioned and stored as separate columns. Each vertical partition may have different or the same organization of data. In Fig. 6, HBase (Apache Software Foundation, 2015) is shown as a column-oriented data store, which ensures a consistent view of the data every time for read/write requests. HBase is an implementation of BigTable (Chang *et al.*, 2008) and developed as a NoSQL database to provide scalable storage and fault-tolerant access to its data. It is deployed on top of Hadoop and HDFS and

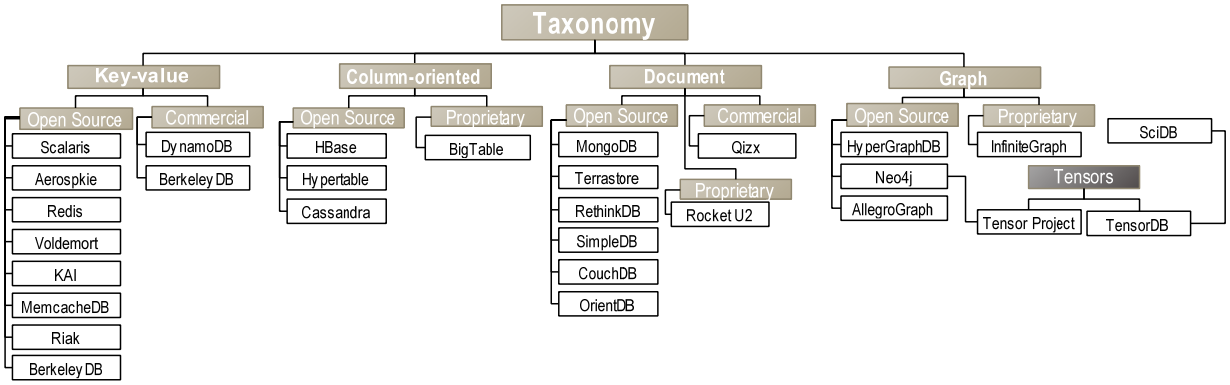

Fig. 6 Taxonomy of big data storage technologies

facilitates efficient random read/write operations. Besides, HBase and BigTable, Hypertable (HyperTable, 2015) and Cassandra (Hewitt, 2010; Lakshman and Malik, 2010) also belong to the column-oriented data model category. MongoDB (MongoDB, 2015), Terrastore (Terrastore, 2015), Rocket U2 (RocketSoftware, 2015), Qizx (Qualcomm, 2014b), RethinkDB (RethinkDB, 2015), OrientDB (OrientDB, 2015), SimpleDB (Sciore, 2007), CouchDB (Apache, 2015), and Riak (Sheehy, 2010) are document-oriented databases. HyperGraphDB (Iordanov, 2010), InfiniteGraph (InfiniteGraph, 2014), Neo4j (Neo4j, 2015), and AllegroGraph (AllegroGraph, 2015) are graph databases. We extend graph-oriented data models and elaborate tensors as analytical models for processing and extracting information from high-dimensional big data. Tensors provide scalability when implemented on distributed systems. Kim and Candan (2014a) have used SciDB (Cudré-Mauroux *et al.*, 2009) to implement tensor decomposition on disk rather than in memory, which minimizes I/O and processing costs. This tensor decomposition project is named TensorDB (Kim and Candan, 2014b) with support of both static and dynamic decompositions. Tensors have also been used in Neo4j.

From the licensing perspective, Fig. 6 shows that most of the available big data storage technologies are open-source and freely available for either a storage solution or a research platform. BigTable is a proprietary storage solution by Google to be built over many commodity servers. Although it is not an open-source project, its implementations, Hypertable and HBase, are available to research and consumer community without any cost. DynamoDB is a commercial storage technology. All storage systems discussed in this paper, except BigTable and DynamoDB, are open-source technologies.

Table 4 explains the taxonomy and describes the types of licensing and data models of each storage system. It also highlights the important features of these storage technologies. The technologies are distributed and are scalable, and they assure fault tolerance and efficient response to read/write operations on the data stored there. Furthermore, the table illustrates the applications of big data storage technologies.

This section has provided an illustration of the taxonomy of big data storage technologies based on the data model and licensing. We have categorized these technologies according to the data model into four NoSQL systems: key-value, column-oriented, document-oriented, and graph databases. The details and applications of each data model are provided. According to the licensing method, storage technologies are categorized as open source, proprietary, and commercial. Fig. 6 shows that most of the storage systems are open-source. However, BigTable (Chang *et al.*, 2008), InfiniteGraph, and Rocket U2 (RocketSoftware, 2015) are proprietary systems, whereas DynamoDB, Qizx (Qualcomm, 2014a), and AllegroGraph are commercial systems. Berkeley DB (Oracle, 2015b) is the only key-value database that is available in both open-source and commercial licenses. In a later section, we analyze NoSQL distributed storage systems according to Brewer's CAP theorem. We discuss the background and importance of Brewer's theorem in analyzing distributed systems. We also list the methods adopted by these storage systems to implement consistency, replication, and partition of data.

Table 4 Taxonomy of big data storage technologies

Data model	License	Technology	Data store/ database*	Features	Applications
Key-value	Open source	Scalaris (Sch *et al*., 2008)	St	Strongly consistent Scalable and highly available with load balancing and fault tolerance Very little maintenance overhead Self-managing	Read/write intensive platforms
		Aerospike (Aerospike, 2015)	St	Highly scalable, consistent, and reliable	User profiles, dynamic web portals, fraud detection SSD cache
		Redis (Carlson, 2013)	St	Automatic partitioning Efficient data read/write access Fault-tolerant and responsive if replica is down	Structured strings LRU cache
		Voldemort (Sumbaly *et al*., 2012; Voldemort, 2015)	St	Automatic data partitioning and replication Transparent fault recovery High read/write availability and horizontal scalability	Large-scale read-only data
		KAI (SourceForge, 2015)	St	Highly fault-tolerant with low latency Scalable with configurable nodes	Web repository Social network
		MemcacheDB (MemcacheDB, 2015)	St	Efficient data storage and retrieval Good performance for read/write High storage availability	Small chunks of arbitrary data (object storage)
		Riak (Sheehy, 2010)	B	Fault-tolerant and highly available Data conflict resolution Support to configure commodity hardware	Unstructured data
	Open source/ commercial	BerkeleyDB (Oracle, 2015b)	B	Scalability and high performance Configurable products Support to complex data management procedures	Embeddable database applications
	Commercial	DynamoDB (Sivasubramanian, 2012)	B	High write availability Automatic replication and fault tolerance	Unstructured variable data
Column-oriented	Open source	HBase	St	Concurrent mode-failure exception hindering read/write performance Auto-split and redistribution	Structured and semi-structured data, random read/write access, social data
		Hypertable (Hypertable, 2015)	B	Highly available and fast random read/write Compatible with many Distributed File Systems	Both structured and unstructured data
		Cassandra (Hewitt, 2010; Lakshman and Malik, 2010)	B	Easy and cost saving High write throughput, no read compromises (locks)	Structured and semi-structured data, time series data, IOT Digg, Rackspace, Raddit, Cloudkick, Twitter, social data
	Proprietary	BigTable (Chang *et al*., 2008)	St	Easy data compression process Fast query response Allowing an infinite number of columns in a table Automatic less reconfiguration to scale the system	Structured large-scale data at Google

* St=data store, B=database

(To be continued)

Table 4

Data model	License	Technology	Data store/ database*	Features	Applications
Document-oriented	Open source	MongoDB (MongoDB, 2015)	B	Easy scalability, fault-tolerance, and high availability Built-in data encryption file system Support to complex schemas Flexibility in data models	Unstructured, dynamic schemas with varying and unknown attributes, social data
		Terrastore (Terrastore, 2015)	St	Persistent in-memory document storage Automatic data redistribution and load balancing Dynamic cluster configuration	Structured big data
		RethinkDB (RethinkDB, 2015)	B	Easy scalability Robustness and more consistency Fast real-time query response	Real-time web/mobile applications
		SimpleDB (Sciore, 2007)	St	High availability Fast query retrieval	Complex queries
		CouchDB (Apache, 2015)	B	Easy use Fault-tolerance Concurrency for request workload	Web application development, social data
		OrientDB (graph database) (OrientDB, 2015)	B	High-speed storage Low-cot scalability Heterogeneity of replicating servers Schema-less	Embedded databases, complex data relationships
	Proprietary	Rocket U2 (RocketSoftware, 2015)	B	Dynamic support to applications Highly efficient, scalable, and reliable for growing data	Business information management
	Commercial	Qizx (Qualcomm, 2014a)	B	Highly scalable, available, and consistent Fast and efficient query execution Support to customized indexing	Semi-structured XML data
Graph	Open source	HyperGraphDB (Iordanov, 2010)	B	Flexible and dynamic schema Non-blocking concurrency Efficient data modeling and knowledge representation	Arbitrary graph data, artificial intelligence and semantic web projects
		Neo4j (Zicari, 2015)	B	Highly scalable and robust Efficient concurrent write transactions without locking Fast for write scaling transaction loads	Intensive data relationship and intensive write transaction applications, social and geospatial data
		AllegroGraph (AllegroGraph, 2015)	B	High storage throughput Highly scalable, available, and persistent Fast query execution and high data load speed	Semantic web reasoning and ontology modeling, complex data
	Commercial	InfiniteGraph (InfiniteGraph, 2014)	B	Rapid development of data-intensive graph-based applications Easy traversal of complex graph elements Processing load distribution Availability of data filtering techniques to improve query performance Easy to use	Growing graph data
SciDB (Cudré-Mauroux *et al.*, 2009)			B	Data update lineage with named versions Data partitioning changing with time Less data loading cost Built-in preprocessing and cleansing functionality Optimized storage and CPU performance	Geospatial, scientific data, astronomy

* St=data store, B=database

4 Analysis of big data storage technologies

In this section we analyze the discussed big data storage technologies. Brewer's CAP theorem, which gives widely used criteria to classify distributed systems, states that it is difficult for NoSQL technologies to fulfill ACID or BASE criteria as consistency, availability, and partition resilience are the considerable factors in designing these large-scale, distributed storage systems. ACID provides a set of characteristics for each transaction performed on relational database management systems and declares that these transactions are fully consistent (strong consistency), whereas BASE allows systems to have a weaker level of consistency. However, for distributed systems, this is not always possible to be consistent. Sometimes, these systems compromise between consistency and availability. Thus, the CAP theorem becomes useful for categorizing modern NoSQL storage technologies. According to new explanation of Brewer's CAP theorem, a distributed system may be either consistent or highly available. Therefore, we explore the characteristics of existing NoSQL technologies and analyze them with Brewer's proposed criteria. Later, we deeply investigate each storage technology to suggest the category of Brewer's theorem where a technology lies. Furthermore, we summarize the mechanisms adopted by modern big data storage technologies to achieve consistency, data partitioning, replication, and indexing.

4.1 Brewer's CAP theorem

We describe the significance of Brewer's CAP theorem in the evaluation of distributed systems. For this purpose, we first describe other criteria and evolution of the CAP theorem. With the emergence of distributed storage systems, the need to reconsider ACID (Gray, 1981) constraints (atomicity, consistency, isolation, and durability) became urging, as it does not ensure availability. Although availability may not be a requirement for some network services, it is most likely to be considerable. Many researchers have presented their viewpoints to explain the basic requirements of a distributed storage system for big data. Fox *et al.* (1997), for example, identified them as incremental scalability, provisioning of overflow growth, high availability, and cost effectiveness for scalable network services. The authors showed that it is very difficult for networked clusters to fulfill all these requirements and presented the BASE (basic availability, soft state, and eventual consistency) model (Fox *et al.*, 1997). This model can be preferably adopted when ACID is not applicable, and it provides a better support to faults and partial failures in clusters at very little cost.

When discussing consistency and availability, it seems that a strongly consistent system prohibits more than one copy of data; thus, the concept of high availability is compromised, and it discourages replication to increase users' access to the required service. Fox and Brewer (1999) introduced a tradeoff to design distributed systems and presented the CAP principle (consistency, availability, and partition resilience). Before explaining the CAP principle and its tradeoff to categorize distributed systems, we provide below a brief definition of the three basic concepts considered related to this principle, namely consistency, availability, and partition resilience. Consistency refers to having the same up-to-date data at each node (Oracle, 2015a), availability suggests quick accessibility to data storage resources with minimum downtime (Oracle, 2015a), and partition resilience is related to fault tolerance in case of unresponsive nodes or a sub-network (Bohlouli *et al.*, 2013).

Initially the authors claimed that a simultaneous provision of these three features as explained above is not possible for a distributed system and two out of three were supposed to be guaranteed (Fox and Brewer, 1999). For instance, there is a possibility that a distributed system compromises partition resilience and satisfies the other two. Existence of any two of these three features can present a distributed system. This principle invited criticism from experts, and, later, Brewer clarified the misleading concept of 'choosing two out of three' and explained that most of the time there is a choice between consistency and availability depending upon the user access pattern or nature of data (Brewer, 2012). As far as partition resilience is concerned, Brewer showed that it is rare to face the partitioning problem in a system and that a lot of options exist for partition handling and recovery. This explanation is depicted in Fig. 7. Thus, the theorem suggests a design of the distributed system be leaning towards either consistency or availability where partition resilience cannot be forfeited.

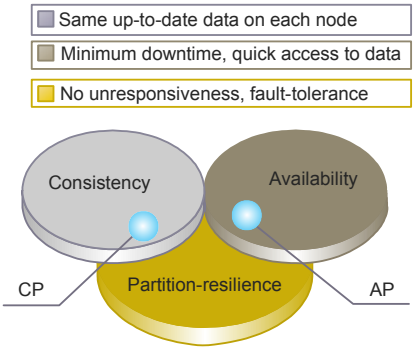

Fig. 7 Brewer's CAP theorem (Brewer, 2012)

4.2 Analysis of big data storage technologies based on Brewer's CAP theorem

We have provided a brief illustration of Brewer's CAP theorem in Section 4.1. According to the new explanation (Brewer, 2012), it is clear that a storage system will be of either CP type or AP type. Here we present a systematic explanation of each storage system available for big data and categorize the system into CP and AP types since it is declared in the CAP theorem that it is impossible for a database to have both ACID consistency and data availability. Consequently, we focus on two pair of combinations from this theorem, which are consistency–partition resilience and availability–partition resilience, in our discussion. Table 5 elaborates the features of big data storage technologies. These features are further considered in analyzing and categorizing these storage systems according to Brewer's theorem. Table 5 also describes the methods for implementing consistency, replication, data partitioning, and indexing for these technologies. Furthermore, Brewer's category is suggested in Table 5 for each storage system.

Based on the investigation of contemporary big data storage technologies presented in Table 5, we suggest the type for each technology as depicted by Brewer. We identify and prove that each technology falls into either the CP or AP category. In Fig. 8 we present the categorization based on the two pairs given by Brewer for modern distributed systems, to show their success and capability towards Brewer's theorem. Intersection of two features presents the criteria, e.g., consistency–partition resilience (CP) and availability–partition resilience (AP). As Fig. 8 describes, recent distributed systems for storing big data satisfy either pair. Consistency and partition resilience are presented in Fig. 8 as CP, and the intersection shows the list of technologies that possess these two features from each data model. The pair availability–partition resilience is also shown in Fig. 8, and at the intersection lie Cassandra, Voldemort, and DynamoDB. We previously explained that, according to Brewer, it is not possible for a system to fully support consistency and availability at the same time. Therefore, for a distributed system it is important to satisfy either choice, but partition resilience is mandatory.

The trivial case for the systems having CP is that these systems become unresponsive to requests until the system recovers from partitioning and data on all sites are up to date. This phenomenon results in a locking state for the system, and the system may become unavailable for this time span. CP-type systems aim to ensure strong consistency regardless of the data access performance, and usually they believe in performing synchronous write operations. Scalaris, Redis, MemcacheDB, and Berkeley DB are CP-type key-value databases.

Scalaris (Schütt *et al.*, 2008) implements a non-blocking Paxos commit protocol to ensure strong consistency. While the master–slave approach of MemcacheDB supports implementation of consistency and clients can access any replica on slaves for read operations, write operations are performed only at the master node.

From the list of column-oriented databases, HBase, Hypertable, and BigTable are of CP type. For up-to-date and consistent data provisioning on all replicas, HBase allows synchronous writes to all replicas, which may increase latency or unavailability, but these are not the design goals for HBase (Wang *et al.*, 2014). HBase is designed to provide read/write consistency and holds clients' requests until the updating operation is complete. There is a write-ahead-log (WAL) in hard drive where write operations are updated first. Later, in-memory data are updated across all replicating nodes so that improved write throughput can be achieved. Similarly, BigTable implements transactional consistency on rows using Chubby service (Burrows, 2006) and the Paxos algorithm (Chandra *et al.*, 2007) for persistent replicas and consistency lock management. Client acquires the lock and maintains the session during which it performs atomic reads and writes.

Table 5 Analysis of big data storage technologies

Data model	License	Technology	Data store/ database[1]	Features	Query language	Consis-tency[2]	Repli-cation[3]	Partition-ing[4]	Index-ing[5]	Brewer's category[6]
Key-value-based	Open source	Scalaris (Sch *et al*., 2008)	St	Strongly consistent Scalable and highly available with load balancing and fault tolerance Very little maintenance overhead Self-managing	Custom	E	Symm,	K	Pr	CP
		Aerospike (Aerospike, 2015)	St	Highly scalable, consistent and reliable	AQL	St	Syn, Asyn	Sh	Sc	AP
		Redis (Carlson, 2013)	St	Automatic partitioning Efficient data read/write access Fault-tolerant and responsive if replica is down	–	E	MS	CH	C	CP
		Voldemort (Sumbaly *et al*., 2012; Voldemort, 2015)	St	Automatic data partitioning and replication Transparent fault recovery High read/write availability and horizontal scalability	Internal	T, E	Sl	CH	C	AP
		KAI (SourceForge, 2015)	St	Highly fault-tolerant with low latency Scalable with configurable nodes	–	E	Asyn	CH	–	AP
		MemcacheDB (MemcacheDB, 2015)	St	Efficient data storage and retrieval Good performance for read/write High storage availability	API	O	MS	None	C	CP
		Riak (Sheehy, 2010)	B	Deriving DynamoDB Fault-tolerant and highly available Data conflict resolution Support to configure commodity hardware	MapReduce	E	MM	CH	Sc	AP
	Open-source/ commercial	BerkeleyDB (Oracle, 2015b)	B	Scalability and high performance Configurable products Support to complex data management procedures	XQuery	E	MS	None	Sc	CP

[1] St=data store, B=database

[2] St=strong, Tr=transactional, T=tunable, E=eventual, O=ordered, PP=per page, Fl=flexible, MV=multi-version, R=runtime

[3] MS=master-slave, MM=multi-master, Sl=selectable replication factor, CR=cross-region, RS=replica set, Syn=synchronous, Asyn=asynchronous, Symm=symmetric

[4] Sh=sharding, CH=consistent hashing, H=horizontal partitioning, K=key-based, Fe=federation, R=runtime

[5] Pr=primary key, Sc=secondary, KV=key-value, C=customized, M=metadata, BT=B-tree, FT=full text

[6] CP=consistency–partition resilience, AP=availability–partition resilience

(To be continued)

Table 5

Data model	License	Technology	Data store/ database[1]	Features	Query language	Consis-tency[2]	Repli-cation[3]	Partition-ing[4]	Index-ing[5]	Brewer's category[6]
	Commercial	DynamoDB (Sivasubramanian, 2012)	B	High write availability Automatic replication and fault tolerance	API	E	CR	CH	Pr, Sc	AP
Column-oriented	Open-source	HBase	St	Concurrent mode failure exception hindering read write performance Auto-split and redistribution	Range, MapReduce	Tr	Sl	Sh	Pr	CP
		Hypertable (Hypertable, 2015)	B	Highly available and fast random read/write Compatible with many distributed file systems	HQL	Tr	Sl	Sh	Sc	CP
		Cassandra (Hewitt, 2010; Lakshman and Malik, 2010)	B	Deriving DynamoDB Easy and cost saving High write throughput, no read compromises (locks)	Range, MapReduce	T, E	Sl	Sh	Sc	AP
	Proprietary	BigTable (Chang *et al*., 2008)	St	Easy data compression process Fast query response Allowing an infinite number of columns in a table Automatic, less recon-figuration to scale the system	GQL	Tr	MS	H	Sc	CP
Document-oriented	Open-source	MongoDB (MongoDB, 2015)	B	Easily scalable, fault-tolerant, and highly available Built-in data encryption file system Support to complex schema Flexibility of data model	SQL, MapReduce	E	RS, MS	Sh	Sc	CP
		Terrastore (Terrastore, 2015)	St	Persistent in-memory document storage Automatic data redistri-bution and load balancing Dynamic cluster configuration	API	PP	MS	CH	C	CP
		RethinkDB (RethinkDB, 2015)	B	Easy scalability Robust and more consistent Fast real-time query response	ReQl	St	Syn, Asyn	Sh	Sc	CP
		SimpleDB (Sciore, 2007)	St	Highly available and fast query retrieval	Erlang	E	MS	Sh	M, C	AP

For description of notes 1–6, see p.1061 (To be continued)

Table 5

Data model	License	Technology	Data store/ database[1]	Features	Query language	Consis-tency[2]	Repli-cation[3]	Partition-ing[4]	Index-ing[5]	Brewer's category[6]
Document-oriented	Open-source	CouchDB (Apache, 2015)	B	Easy to use Fault-tolerant Concurrency for request workload	MapReduce	E	MM	Sh	BT	AP
		OrientDB (OrientDB, 2015)	B	High-speed storage Low-cot scalability Heterogeneity of replicating servers Schema-less	SQL	MV	MM	Sh	Sc	AP
	Proprietary	Rocket U2 (RocketSoftware, 2015)	B	Dynamic support to applications Highly efficient, scalable, and reliable for growing data	RetrieVe and UniQuery	R	MS, Asyn	None	Sc	AP
	Commercial	Qizx (Qualcomm, 2014a)	B	Highly scalable, available, and consistent Fast and efficient query execution Support to customized indexing	XQuery, REST	Tr	MS	R	FT, C	AP
Graph-based	Open-source	HyperGraphDB (Iordanov, 2010)	B	Flexible and dynamic schema Non-blocking concurrency Efficient data modeling and knowledge representation	API	E	P2P	P2P	KV	CP
		Neo4j	B	Highly scalable and robust Efficient concurrent write transactions without locking Fast for write scaling transaction loads	SPARQL, internal	St, E	MS	Sh	Sc	CP
		AllegroGraph (AllegroGraph, 2015)	B	High storage throughput Highly scalable, available, and persistent Fast query execution and high data load speed	SPARQL, RDFS++	Tr	MS	Fe	Sc, Sp	CP
	Commercial	InfiniteGraph (InfiniteGraph, 2014)	B	Rapid development of data-intensive, graph-based applications Easy traversal of complex graph elements Processing load distribution Availability of data filtering techniques to improve query performance Easy to use	Internal, API	Fl, E	Syn	Sh	Sc	CP

For description of notes 1–6, see p.1061

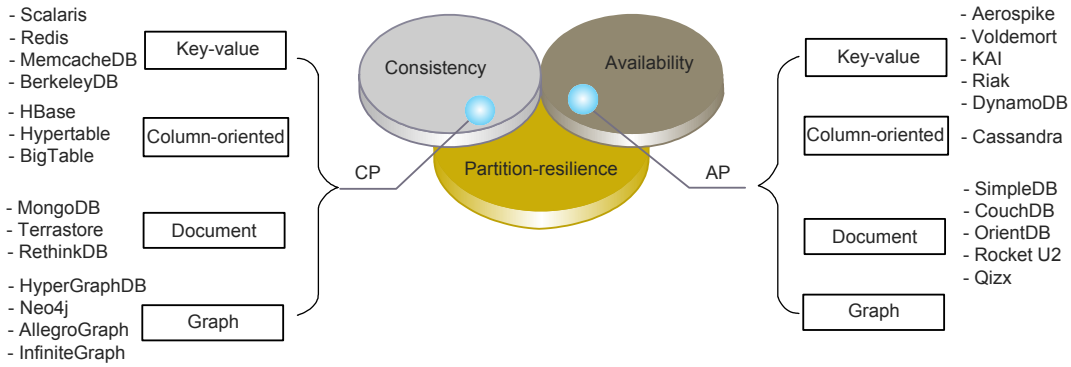

Fig. 8 Big data storage technologies and the CAP theorem

Document databases such as MongoDB, Terrastore, and RethinkDB are CP-type databases. These databases prefer consistency over availability. MongoDB and RethinkDB are authoritative systems from the design perspective. Authoritative systems do not face the data inconsistency problem, which makes MongoDB and RethinkDB simple to build applications on top of them, whereas unavailability occurs very often for them (RethinkDB, 2015). Terrastore ensures that each single document is consistent and up to date. Because Terrastore does not allow inter-document transactions or multiple simultaneous operations on the same document, Terrastore becomes more scalable with simpler consistency. Tolerance to node failure is provided by Terrastore to the extent that at least one server node is working and the server node must have connection to the master node. Otherwise, Terrastore will not ensure partition resilience.

All the graph databases discussed in this paper are of CP type. None of them is more inclined to support availability than consistency. AllegroGraph is the only graph database that implements transactional consistency; the rest of the graph databases aim to provide eventual consistency. HyperGraphDB implements Berkeley DB and offers non-blocking concurrency for transactions. Furthermore, HyperGraphDB has flexible schema, and updates are eventually consistent. Similarly, Neo4j supports non-blocking concurrency and eventual consistency. Even write-intensive transactions are efficiently accomplished by Neo4j. InfiniteGraph also supports eventual consistency for write operations.

High availability to its extent can be achieved only when each request has received a response and the system does not ever face a complete locking state. However, it will not guarantee consistent results of requests. Key-value databases such as Aerospike, Voldemort, KAI, Riak, and DynamoDB are AP-type databases. Voldemort is designed as a highly available database where eventual consistency ensures almost up-to-date replicas. DynamoDB, when receiving a data update request, does not propagate it to all replicas for the reason of not locking them from availability to read requests. DynamoDB performs updating on a quorum of servers, while other servers respond to clients' requests in order to provide them access to consistent data at that time (Ramakrishnan, 2012). The quorum of servers needs to ensure successful write update before any request is entertained (Wang *et al.*, 2014). Riak has complex conflict resolution strategies, and implementation of quorum also results in latency problems. However, Riak is strongly available and is a partition-resilient key-value system.

Cassandra is the only column-oriented AP-type database. Cassandra adopts the same mechanism to ensure availability and propagate consistency as DynamoDB does. Like Riak, Cassandra also has complex procedures for conflict resolution and quorum implementation. SimpleDB, CouchDB, OrientDB, Rocket U2, and Qizx are AP-type document databases. OrientDB offers versioning to implement the effect of write operations. In this way, effect of change in data can be viewed in different versions. As far as graph databases discussed in this paper are concerned, none of them is of AP type. In the next section we will summarize the discussion and identify future research challenges.

5 Discussion and future research challenges

This paper has presented the evolution and taxonomy of big data storage technologies. Technologies are commonly designed to provide a storage solution along with high scalability to growing data volumes with heterogeneous data structures. These databases are deployed over distributed systems to achieve high availability, improved data access, performance, and fault tolerance. However, the extent of provisioning these services is different for each database, which makes these databases distinguishable from each other. This extent helps identify these databases as of CP or AP type. In Table 5, contemporary key-value, column-oriented, document-oriented, and graph databases are presented and analyzed on the basis of their adopted procedures for consistency, data partitioning, replication, and indexing. Moreover, their correlation with Brewer's CAP categorization is kept in the analysis to suggest their type.

Key-value databases apply data partitioning on separate records regardless of each having the same attributes. A unique key is assigned to each record, and the value contains data of a record. Although they are mostly suitable for unstructured data, structured data can be presented with these databases if record-based data retrieval and analysis needs to be performed on it. As far as licensing is concerned, most of the column-oriented and key-value databases are open source. BigTable is the only column-oriented database that has proprietary license. Likewise, DynamoDB is the only key-value database that is available commercially. Column-oriented databases such as HBase, Hypertable, Cassandra, and BigTable are mostly suitable for structured data with enough support to unstructured data. These databases apply vertical partitioning on the data and store each column as a separate chunk of data, and performing queries on attributes as well as attribute-based analysis of data is easier with this data model.

As far as document-oriented databases are concerned, these databases also have the key-value data structure. However, the value identifies a document instead of a record. Documents are usually XML files with some schema. Compared to the key-value data model, document databases have less support to scalability and unstructured data. Furthermore, databases having document structure are, on average, prone to availability and consistency. For instance, MongoDB, Terrastore, and RethinkDB are consistent databases, whereas SimpleDB, CouchDB, OrientDB, Rocket U2, and Qizx are highly available. Graph databases are well-structured databases, where analyzing data as well as their relationships is significant. Although graph databases do not have good support to scalability and clustering, these databases offer complex data structures. According to our analysis, all graph databases in this survey are CP-type systems.

With the proliferation of big data, industry and academia are more interested in data management than computational management. The technology has much evolved in provisioning vast storage and processing resources. However, in big data management, efficient techniques for data acquisition, preprocessing, processing, and storage are desirable. The ongoing development is focusing on bringing efficient solutions that support big data management. The Hadoop framework (Lam *et al*., 2010) has become the de facto standard for big data processing with the MapReduce programming model, which offers batch processing of extensive volume of files residing over commodity hardware, whereas for real-time big data processing, the simple, scalable streaming system (S4) (Neumeyer *et al.*, 2010) is a widely adopted tool. The Apache Software Foundation has a list of contributions to big data solutions such as Mahout, Lucene, Hive, Pig, and Spark.

Besides processing, storage optimization is also important. Methods for data clustering, replication, and indexing for efficient storage utilization and data retrieval are of main concern. Storage-optimizing hierarchical agglomerative clustering (Buza *et al.*, 2014), the K-means algorithm (Zhao *et al.*, 2009), and the artificial bee colony (ABC) algorithm (Karaboga and Ozturk, 2011) are the clustering approaches used in recent research. The storage technologies presented in this paper have built-in support to replication, which in turn ensures data availability, fault tolerance, and less data accessing delay. For efficient replication, the ABC algorithm (Taheri *et al.*, 2013), D2RS (Sun *et al.*, 2012), and JXTA-overlay P2P platform (Spaho *et al.*, 2013) are the famous techniques used. HAIL (Dittrich *et al.*, 2012) provides an indexing solution for Hadoop, which improves the data search and retrieval process.

While summarizing the storage technologies presented in this paper, it can be stated that all the storage structures are partition-resilient, meaning that network partitioning and disconnections in distributed systems is rare and there are many options to handle and recover from a partitioning situation. Thus, the choice is only between consistency and availability. As illustrated in Section 4, the support to both of them is beyond the possibility of a distributed system. Thus, the discussed distributed storage systems are placed in either category. Storage systems that provide more support to consistency are CP-type systems. Apart from the efficiency of available big data storage technologies, the challenges of storing future big data still need to be addressed and considered. Finding and adopting the tradeoff between consistency and availability so that more throughput gains can be achieved from distributed storage systems leads to a number of challenges in this research. The following are some of these challenges:

1. Frequent data update and schema change: The update rate is mostly very high and the volume of data is growing very rapidly. In case of unstructured data, change in schema is also very common. However, available storage technologies are scalable, but the need to be efficient in data updates and schema is still under consideration. Some technologies like HDFS do not offer data update but append operation in only support.

2. Partitioning method: Data models suggest two methods of big data partitioning to make it contained on distributed storage nodes accordingly. Horizontal or vertical partitioning is applied on data based on access patterns. Data may be required to be analyzed by the features or records. Thus, the choice of column-oriented or key-value NoSQL databases is available. However, the prediction of access pattern might be wrong or change during execution. This poses a critical research challenge on existing data models specified for big data storage solutions.

3. Replication factor: Data are replicated over multiple sites to achieve fault tolerance and high availability to its users. Although this concept makes the storage very efficient to improve access performance, it compromises data consistency and does not suit in frequent data changing and up-to-date access requirement conditions. This leads to poorer access performance if frequent consistency locks are experienced. Furthermore, the number of replicas for data is a multiple of storage space consumption. Sometimes it looks wise to access data from a remote site rather to use local storage space. Therefore, preconfigured or customized replication by applications or users is a challenge.

4. User expertise: Data are becoming more complex nowadays. At the same time, the user space is broadened by enterprises, so that users from different domains can execute queries on data according to their problems. It reveals the requirement of simple-to-deploy and easy-to-use storage but with higher performance than the relational database solution. To achieve improved performance, sometimes these databases integrate DBMS platforms, which undoubtedly meets the expectations, but the implementation and configuration process becomes complex for non-expert users.

6 Conclusions

A categorization of contemporary storage technologies for big data has been attempted in this paper. The main objective of the paper is to investigate and analyze state-of-the-art big data storage technologies and to present their categorization based on Brewer's CAP theorem for distributed systems, so that researchers and big data analysts can explore enhanced storage solutions in specific domains, where availability, consistency, and fault tolerance are considerable requirements. Moreover, presentation of limitations of existing storage technologies for adequate scalability is a major concern of this work. This paper focuses mainly on categorization under data models and classifies modern storage technologies for big data as key-value, column-oriented, document-oriented, and graph systems. These technologies are further studied and analyzed by a critical review using Brewer's CAP theorem. A discussion is carried out to survey and highlight the efficiency of each storage system for the scalability, consistency, and availability requirements of big data. Future key challenges with regard to storing big data are also emphasized in the discussion. In conclusion, it can be stated that systems are inclined towards provisioning consistency and availability as required by applications or users. This leading phenomenon is used to suggest

categorization of available big data storage systems so that the selection with either preference becomes obvious.

References

Aasman, J., 2008. Event Processing Using an RDF Database (White Paper). Association for the Advancement of Artificial Intelligence, p.1-5.

Abadi, D.J., Boncz, P.A., Harizopoulos, S., 2009. Column-oriented database systems. *Proc. VLDB Endow.*, **2**(2): 1664-1665. https://doi.org/10.14778/1687553.1687625

Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., *et al*., 2009. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proc. VLDB Endow*., **2**(1):922-933. https://doi.org/10.14778/1687627.1687731

Abramova, V., Bernardino, J., 2013. NoSQL databases: MongoDB vs Cassandra. Proc. Int. Conf. on Computer Science and Software Engineering, p.14-22. https://doi.org/10.1145/2494444.2494447

Aerospike, 2012. Aerospike, Architecture Overview (White Paper). http://www.aerospike.com/

Aerospike, 2015. NoSQL Database, In-Memory or Flash Optimized and Web Scale. http://www.aerospike.com/ [Accessed on May 5, 2015].

Alex, P., Ana-Maria, B., 2009. Terrastore: a Consistent, Partitioned and Elastic Document Database. http://nosql.mypopescu.com/post/304908601/terrastore-consistent-partitioned-elastic-document-datab [Accessed on May 7, 2015].

AllegroGraph, 2015. AllegroGraph. http://franz.com/agraph/allegrograph/ [Accessed on May 5, 2015].

Anderson, J.C., Lehnardt, J., Slater, N., 2010. CouchDB: the Definitive Guide. O'Reilly Media, Inc., California.

Apache, 2015. Apache CouchDB: a Database for the Web. http://couchdb.apache.org/ [Accessed on May 5, 2015].

Apache Software Foundation, 2015. HBase Apache. http://hbase.apache.org/ [Accessed on Jan. 15, 2015].

Armbrust, M., Fox, A., Patterson, D., *et al*., 2009. Scads: scale-independent storage for social computing applications. arXiv:0909.1775.

Azeem, R., Khan, M.I.A., 2012. Techniques about data replication for mobile ad-hoc network databases. *Int. J. Multidiscipl. Sci. Eng.*, **3**(5):53-57.

Banker, K., 2011. MongoDB in Action. Manning Publications Co., New York.

Baron, J., Kotecha, S., 2013. Storage Options in the AWS Cloud. Technical Report, Amazon Web Services, Washington DC.

Batra, S., Tyagi, C., 2012. Comparative analysis of relational and graph databases. *Int. J. Soft Comput. Eng.*, **2**(2):509-512.

Bohlouli, M., Schulz, F., Angelis, L., *et al*., 2013. Towards an integrated platform for big data analysis. *In*: Fathi, M. (Ed.), Integration of Practice-Oriented Knowledge Technology: Trends and Prospectives. Springer Berlin Heidelberg, p.47-56. https://doi.org/10.1007/978-3-642-34471-8_4

Borthakur, D., 2008. HDFS Architecture Guide. http://hadoop.apache.org/common/docs/current/hdfsdesign.pdf

Bossa, S., 2009. Thoughts and Fragments: Terrastore and the CAP Theorem. http://sbtourist.blogspot.com/2009/12/terrastore-and-cap-theorem.html [Accessed on May 7, 2015].

Brewer, E., 2012. CAP twelve years later: how the "rules" have changed. *Computer*, **45**(2):23-29. https://doi.org/10.1109/MC.2012.37

Bunch, C., Chohan, N., Krintz, C., *et al*., 2010. An evaluation of distributed datastores using the AppScale cloud platform. IEEE 3rd Int. Conf. on Cloud Computing, p.305-312. https://doi.org/10.1109/CLOUD.2010.51

Burrows, M., 2006. The Chubby lock service for loosely-coupled distributed systems. Proc. 7th Symp. on Operating Systems Design and Implementation, p.335-350.

Buza, K., Nagy, G.I., Nanopoulos, A., 2014. Storage-optimizing clustering algorithms for high-dimensional tick data. *Expert Syst. Appl.*, **41**(9):4148-4157. https://doi.org/10.1016/j.eswa.2013.12.046

Carlson, J., 2013. Redis in Action. Manning Publications Co., New York.

Cattell, R., 2010. Scalable SQL and NoSQL data stores. *SIGMOD Rec*., **39**(4):12-27. https://doi.org/10.1145/1978915.1978919

Chandra, T.D., Griesemer, R., Redstone, J., 2007. Paxos made live: an engineering perspective. Proc. 26th Annual ACM Symp. on Principles of Distributed Computing, p.398-407. https://doi.org/10.1145/1281100.1281103

Chang, F., Dean, J., Ghemawat, S., *et al*., 2008. Bigtable: a distributed storage system for structured data. *ACM Trans. Comput. Syst*., **26**(2):1-26. https://doi.org/10.1145/1365815.1365816

Chen, C.L.P., Zhang, C.Y., 2014. Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inform. Sci.*, **275**:314-347. https://doi.org/10.1016/j.ins.2014.01.015

Chen, M., Mao, S.W., Liu, Y.H., 2014. Big data: a survey. *Mob. Networks Appl.*, **19**(2):171-209. https://doi.org/10.1007/s11036-013-0489-0

Cichocki, A., 2014. Era of big data processing: a new approach via tensor networks and tensor decompositions. arXiv:1403.2048.

Coburn, J., Caulfield, A.M., Akel, A., *et al*., 2011. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. *ACM SIGPLAN Not*., **47**(3):105-118. https://doi.org/10.1145/2248487.1950380

Cudré-Mauroux, P., Kimura, H., Lim, K.T., *et al*., 2009. A demonstration of SciDB: a science-oriented DBMS. *Proc. VLDB Endow.*, **2**(2):1534-1537. https://doi.org/10.14778/1687553.1687584

Deagustini, C.A.D., Dalibón, S.E.F., Gottifredi, S., *et al*., 2013. Relational databases as a massive information source for defeasible argumentation. *Knowl.-Based Syst.*, **51**:93-109. https://doi.org/10.1016/j.knosys.2013.07.010

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM*, **51**(1):107-113. https://doi.org/10.1145/1327452.1327492

DeCandia, G., Hastorun, D., Jampani, M., *et al*., 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Oper. Syst. Rev.*, **41**(6):205-220. https://doi.org/10.1145/1323293.1294281

Deka, G.C., 2014. A survey of cloud database systems. *IT Prof.*, **16**(2):50-57. https://doi.org/10.1109/MITP.2013.1

Dharavath, R., Kumar, C., 2015. A scalable generic transaction model scenario for distributed NoSQL databases. *J. Syst. Softw.*, **101**:43-58. https://doi.org/10.1016/j.jss.2014.11.037

Diack, B.W., Ndiaye, S., Slimani, Y., 2013. CAP theorem between claims and misunderstandings: what is to be sacrificed? *Int. J. Adv. Sci. Technol.*, **56**:1-12.

Dittrich, J., Quiané-Ruiz, J., Richter, S., *et al*., 2012. Only aggressive elephants are fast elephants. *Proc. VLDB Endow.*, **5(11)**:1591-1602. https://doi.org/10.14778/2350229.2350272

Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vañó, A., *et al*., 2010. Survey of graph database performance on the HPC scalable graph analysis benchmark. *In*: Shen, H.T., Pei, J., Özsu, M.T., *et al*. (Eds.), Web-Age Information Management. Springer Berlin Heidelberg, p.37-48. https://doi.org/10.1007/978-3-642-16720-1_4

Excoffier, L., Lischer, H.E.L., 2010. Arlequin suite ver 3.5: a new series of programs to perform population genetics analyses under Linux and Windows. *Mol. Ecol. Res.*, **10**(3):564-567. https://doi.org/10.1111/j.1755-0998.2010.02847.x

Fox, A., Brewer, E.A., 1999. Harvest, yield, and scalable tolerant systems. Proc. 7th Workshop on Hot Topics in Operating Systems, p.174-178. https://doi.org/10.1109/HOTOS.1999.798396

Fox, A., Gribble, S.D., Chawathe, Y., *et al*., 1997. Cluster-based scalable network services. Proc. 16th ACM Symp. on Operating Systems Principles, p.78-91. https://doi.org/10.1145/268998.266662

Fulton, S., 2011. The Other Non-SQL Alternative: Infinite Graph 2.0. http://readwrite.com/2011/08/24/the-other-non-sql-alternative [Accessed on May 5, 2015].

Gani, A., Siddiqa, A., Shamshirband, S., *et al*., 2015. A survey on indexing techniques for big data: taxonomy and performance evaluation. *Knowl. Inform. Syst.*, **46**(2):241-284. https://doi.org/10.1007/s10115-015-0830-y

George, L., 2011. HBase: the Definitive Guide. O'Reilly Media, Inc., California.

Ghemawat, S., Gobioff, H., Leung, S.T., 2003. The Google file system. *SIGOPS Oper. Syst. Rev.*, **37**(5):29-43. https://doi.org/10.1145/1165389.945450

Gorton, I., Klein, J., 2015. Distribution, data, deployment: software architecture convergence in big data systems. *IEEE Softw.*, **32**(3):78-85. https://doi.org/10.1109/MS.2014.51

Gray, J., 1981. The transaction concept: virtues and limitations. Proc. 7th Int. Conf. on Very Large Data Bases, p.144-154.

Habeeb, M., 2010. A Developer's Guide to Amazon SimpleDB. Addison-Wesley Professional.

Han, J., Haihong, E., Le, G., *et al*., 2011. Survey on NoSQL database. 6th Int. Conf. on Pervasive Computing and Applications, p.363-366. https://doi.org/10.1109/ICPCA.2011.6106531

Hecht, R., Jablonski, S., 2011. NoSQL evaluation: a use case oriented survey. Int. Conf. on Cloud and Service Computing, p.336-341. https://doi.org/10.1109/CSC.2011.6138544

Helmke, M., 2012. Ubuntu Unleashed 2012 Edition: Covering 11.10 and 12.04. Sams Publishing.

Hewitt, E., 2010. Cassandra: the Definitive Guide. O'Reilly Media, Inc., California.

Hilker, S., 2012. Survey Distributed Databases—Toad for Cloud. http://www.toadworld.com/products/toad-for-cloud-databases/w/wiki/308.survey-distributed-databases.aspx [Accessed on May 5, 2015].

Hu, Y., Dessloch, S., 2014. Extracting deltas from column oriented NoSQL databases for different incremental applications and diverse data targets. *Data Knowl. Eng.*, **93**:42-59. https://doi.org/10.1016/j.datak.2014.07.002

HyperGraphDB, 2010. HyperGraphDB—A Graph Database. http://www.hypergraphdb.org/ [Accessed on May 5, 2015].

HyperTable, 2015. Hypertable. http://hypertable.com/documentation/ [Accessed on Jan. 15, 2015].

IMDB, 2015. The Internet Movie DataBase. http://www.imdb.com/ [Accessed on May 5, 2015].

InfiniteGraph, 2014. InfiniteGraph | Distributed Graph Database. http://www.objectivity.com/ [Accessed on May 5, 2015].

Iordanov, B., 2010. HyperGraphDB: a generalized graph database. *In*: Shen, H.T., Pei, J., Özsu, M.T., *et al*. (Eds.), Web-Age Information Management. Springer Berlin Heidelberg, p.25-36. https://doi.org/10.1007/978-3-642-16720-1_3

Kaisler, S., Armour, F., Espinosa, J.A., *et al*., 2013. Big data: issues and challenges moving forward. 46th Hawaii Int. Conf. on System Sciences, p.995-1004. https://doi.org/10.1109/HICSS.2013.645

Karaboga, D., Ozturk, C., 2011. A novel clustering approach: artificial bee colony (ABC) algorithm. *Appl. Soft Comput.*, **11**(1):652-657. https://doi.org/10.1016/j.asoc.2009.12.025

Khetrapal, A., Ganesh, V., 2006. HBase and Hypertable for Large Scale Distributed Storage Systems. Department of Computer Science, Purdue University.

Kim, M., Candan, K.S., 2014a. Efficient static and dynamic in-database tensor decompositions on chunk-based array stores. Proc. 23rd ACM Int. Conf. on Information and Knowledge Management, p.969-978. https://doi.org/10.1145/2661829.2661864

Kim, M., Candan, K.S., 2014b. TensorDB: in-database tensor manipulation with tensor-relational query plans. Proc. 23rd ACM Int. Conf. on Information and Knowledge Management, p.2039-2041. https://doi.org/10.1145/2661829.2661842

Kristina, C., Michael, D., 2010. MongoDB: the Definitive Guide. O'Reilly Media, Inc., California.

Kumar, G., 2014. Just Say Yes to NoSQL. http://www.3pillarglobal.com/insights/just-say-yes-to-nosql [Accessed on May 5, 2015].

Lakshman, A., Malik, P., 2010. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, **44**(2): 35-40. https://doi.org/10.1145/1773912.1773922

Lam, C.F., Liu, H., Koley, B., *et al*., 2010. Fiber optic communication technologies: what's needed for datacenter network operations. *IEEE Commun. Mag.*, **48**(7):32-39. https://doi.org/10.1109/MCOM.2010.5496876

Lorica, B., 2015. The Tensor Renaissance in Data Science. http://radar.oreilly.com/2015/05/the-tensor-renaissance-in-data-science.html

MacFadden, G., 2013. 21 NoSQL Innovators to Look for in 2020. http://blog.parityresearch.com/21-nosql-innovators-to-look-for-in-2020/ [Accessed on May 5, 2015].

MemcacheDB, 2015. MemcacheDB. http://memcachedb.org/ [Accessed on Jan. 10, 2015].

Milne, D., Witten, I.H., 2013. An open-source toolkit for mining Wikipedia. *Artif. Intell.*, **194**:222-239. https://doi.org/10.1016/j.artint.2012.06.007

MongoDB, 2015. MongoDB Architecture Guide (White Paper). https://www.mongodb.com/lp/white-paper/architecture-guide?jmp=docs&_ga=1.165918654.1239465962.1430978187:MongoDB [Accessed on May 7, 2015].

Montag, D., 2013. Understanding Neo4j Scalability. http://info.neotechnology.com/rs/neotechnology/images/Understanding%20Neo4j%20Scalability(2).pdf

MovieLens, 2015. MovieLens. https://movielens.org/ [Accessed on May 5, 2015].

Nagy, D., Yassin, A.M., Bhattacherjee, A., 2010. Organizational adoption of open source software: barriers and remedies. *Commun. ACM*, **53**(3):148-151. https://doi.org/10.1145/1666420.1666457

Neo4j, 2015. Overcoming SQL Strain and SQL Pain (White Paper). http://neo4j.com/resources/wp-overcoming-sql-strain/?utm_source=db-engines&utm_medium=textsqlpain&utm_content=download&utm_campaign=dl [Accessed on May 5, 2015].

Neumeyer, L., Robbins, B., Nair, A., *et al*., 2010. S4: distributed stream computing platform. IEEE Int. Conf. on Data Mining Workshops, p.170-177. https://doi.org/10.1109/ICDMW.2010.172

Niranjanamurthy, M., Archana, U.L., Niveditha, K.T., *et al*., 2014. The research study on DynamoDB—NoSQL database service. *Int. J. Comput. Sci. Mob. Comput.*, **3**(10):268-279.

Objectivity, Inc., 2012. InfiniteGraph: the Distributed Graph Database (White Paper). http://www.objectivity.com/products/infinitegraph/ [Accessed on May 5, 2015].

Oliveira, S.F., Fürlinger, K., Kranzlmüller, D., 2012. Trends in computation, communication and storage and the consequences for data-intensive science. IEEE 14th Int. Conf. on High Performance Computing and Communication and IEEE 9th Int. Conf. on Embedded Software and Systems, p.572-579. https://doi.org/10.1109/HPCC.2012.83

Oracle, 2015a. Managing Consistency with Berkeley DB-HA (White Paper). http://www.oracle.com/technetwork/products/berkeleydb/high-availability-099050.html [Accessed on May 5, 2015].

Oracle, 2015b. Oracle Berkeley DB. http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html [Accessed on May 5, 2015].

Oracle, 2015c. Unleash High Availability Applications with Berkeley DB (White Paper). http://www.oracle.com/technetwork/products/berkeleydb/high-availability-099050.html [Accessed on May 5, 2015].

Oracle Secondary, 2015. Secondary Indexes. https://docs.oracle.com/cd/E17275_01/html/programmer_reference/am_second.html [Accessed on May 5, 2015].

OrientDB, 2015. OrientDB—OrientDB Mulit-model NoSQL Database. http://orientdb.com [Accessed on May 5, 2015].

Padhye, V., Tripathi, A., 2015. Scalable transaction management with snapshot isolation for NoSQL data storage systems. *IEEE Trans. Serv. Comput.*, **8**(1):121-135. https://doi.org/10.1109/TSC.2013.47

Pokorny, J., 2013. NoSQL databases: a step to database scalability in web environment. *Int. J. Web Inform. Syst.*, **9**(1):69-82. https://doi.org/10.1108/17440081311316398

Putnik, G., Sluga, A., ElMaraghy, H., *et al*., 2013. Scalability in manufacturing systems design and operation: state-of-the-art and future developments roadmap. *CIRP Ann. Manuf. Technol.*, **62**(2):751-774. https://doi.org/10.1016/j.cirp.2013.05.002

Qualcomm, 2014a. NoSQL XML Databases Qualcomm Qizx. https://www.qualcomm.com/qizx [Accessed on May 5, 2015].

Qualcomm, 2014b. Qualcomm Qizx | User Guide. https://www.qualcomm.com/qizx [Accessed on May 5, 2015].

Ramakrishnan, R., 2012. CAP and cloud data management. *Computer*, **45**(2):43-49. https://doi.org/10.1109/MC.2011.388

RethinkDB, 2015. RethinkDB: the Open Source Database for Real-Time Web. http://rethinkdb.com/ [Accessed on May 5, 2015].

RocketSoftware, 2014a. High Availability and Dissaster Recovery for Rocket U2 Databases. http://info.rocketsoftware.com/hadr.html [Accessed on May 5, 2015].

RocketSoftware, 2014b. Vermont Teddy Bear | A Top Ecommerce Retailer Relies on Rocket U2 to Successfully Manage Information Processing Activities in Its Direct-to-Consumer Divisions [Case Study]. http://blog.rocketsoftware.com/blog/2014/12/22/vermont-teddy-bear-relies-rocket-u2-improve-service-increase-revenue/

RocketSoftware, 2015. Flexible, High Volume Data Management | Rocket Software. http://www.rocketsoftware.com/product-families/rocket-u2 [Accessed on May 5, 2015].

Ruflin, N., Burkhart, H., Rizzotti, S., 2011. Social-data storage-systems. Proc. Databases and Social Networks, p.7-12. https://doi.org/10.1145/1996413.1996415

Sakr, S., Liu, A., Batista, D.M., *et al*., 2011. A survey of large scale data management approaches in cloud environments. *IEEE Commun. Surv. Tutor.*, **13**(3):311-336. https://doi.org/10.1109/SURV.2011.032211.00087

Scalaris, 2015. Scalaris. http://scalaris.zib.de/ [Accessed on May 5, 2015].

Schütt, T., Schintke, F., Reinefeld, A., 2008. Scalaris: reliable transactional P2P key/value store. Proc. 7th ACM

SIGPLAN Workshop on ERLANG, p.41-48. https://doi.org/10.1145/1411273.1411280

Sciore, E., 2007. SimpleDB: a simple Java-based multiuser system for teaching database internals. *SIGCSE Bull.*, **39**(1):561-565. https://doi.org/10.1145/1227504.1227498

SD Times Newswire, 2013. OrientDB Becomes Distributed Using Hazelcast, Leading Open Source In-Memory Data Grid. http://sdtimes.com/orientdb-becomes-distributed-using-hazelcast-leading-open-source-in-memory-data-grid/ [Accessed on May 5, 2015].

Seltzer, M., Bostic, K., 2015. The Architecture of Open Source Applications: Berkeley DB. http://www.aosabook.org/en/bdb.html [Accessed on May 5, 2015].

Sheehy, J., 2010. Riak 0.10 is Full of Great Stuff. http://basho.com/riak-0-10-is-full-of-great-stuff/ [Accessed on May 5, 2015].

Shvachko, K.V., 2010. HDFS scalability: the limits to growth. *Login*, **35**(2):6-16.

Sivasubramanian, S., 2012. Amazon DynamoDB: a seamlessly scalable non-relational database service. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.729-730. https://doi.org/10.1145/2213836.2213945

Skoulis, I., Vassiliadis, P., Zarras, A.V., 2015. Growing up with stability: how open-source relational databases evolve. *Inform. Syst.*, **53**:363-385. https://doi.org/10.1016/j.is.2015.03.009

SourceForge, 2015. KAI SourceForge. http://sourceforge.net/projects/kai/ [Accessed on May 5, 2015].

Spaho, E., Barolli, L., Xhafa, F., *et al*., 2013. P2P data replication and trustworthiness for a JXTA-overlay P2P system using fuzzy logic. *Appl. Soft Comput.*, **13**(1):321-328. https://doi.org/10.1016/j.asoc.2012.08.044

Stonebraker, M., Brown, P., Zhang, D., *et al*., 2013. SciDB: a database management system for applications with complex analytics. *Comput. Sci. Eng.*, **15**(3):54-62. https://doi.org/10.1109/MCSE.2013.19

Subramaniyaswamy, V., Vijayakumar, V., Logesh, R., *et al*., 2015. Unstructured data analysis on big data using MapReduce. *Proc. Comput. Sci.*, **50**:456-465. https://doi.org/10.1016/j.procs.2015.04.015

Sumbaly, R., Kreps, J., Gao, L., *et al*., 2012. Serving large-scale batch computed data with project Voldemort. Proc. 10th USENIX Conf. on File and Storage Technologies, p.18.

Sun, D.W., Chang, G.R., Gao, S., *et al*., 2012. Modeling a dynamic data replication strategy to increase system availability in cloud computing environments. *J. Comput. Sci. Technol.*, **27**(2):256-272. https://doi.org/10.1007/s11390-012-1221-4

Taheri, J., Lee, Y.C., Zomaya, A.Y., *et al*., 2013. A bee colony based optimization approach for simultaneous job scheduling and data replication in grid environments. *Comput. Oper. Res.*, **40**(6):1564-1578. https://doi.org/10.1016/j.cor.2011.11.012

Tanenbaum, A., van Steen, M., 2007. Distributed Systems. Pearson Prentice Hall.

Taylor, R.C., 2010. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinform.*, **11**(Suppl 12):1-6. https://doi.org/10.1186/1471-2105-11-S12-S1

Terrastore, 2015. Terrastore—Scalable, Elastic, Consistent Document Store. http://code.google.com/p/terrastore/ [Accessed on May 7, 2015].

Tudorica, B.G., Bucur, C., 2011. A comparison between several NoSQL databases with comments and notes. 10th Roedunet Int. Conf., p.1-5. https://doi.org/10.1109/RoEduNet.2011.5993686

Turk, A., Selvitopi, R.O., Ferhatosmanoglu, H., *et al*., 2014. Temporal workload-aware replicated partitioning for social networks. *IEEE Trans. Knowl. Data Eng.*, **26**(11):2832-2845. https://doi.org/10.1109/TKDE.2014.2302291

Vicknair, C., Macias, M., Zhao, Z.D., *et al*., 2010. A comparison of a graph database and a relational database: a data provenance perspective. Proc. 48th Annual Southeast Regional Conf., p.1-6. https://doi.org/10.1145/1900008.1900067

Voldemort, 2015. Project Voldemort. http://www.project-voldemort.com/voldemort/ [Accessed on Jan. 10, 2015].

Vyas, U., Kuppusamy, P., 2014. DynamoDB Applied Design Patterns. Packt Publishing Ltd., Birmingham.

Walsh, L., Akhmechet, V., Glukhovsky, M., 2009. RethinkDB-Rethinking Database Storage (White Paper).

Wang, H.J., Li, J.H., Zhang, H.M., *et al*., 2014. Benchmarking Replication and Consistency Strategies in Cloud Serving Databases: HBase and Cassandra. *In*: Zhan, J.F., Han, R., Weng, C.L. (Eds.), Big Data Benchmarks, Performance Optimization, and Emerging Hardware. Springer International Publishing, p.71-82. https://doi.org/10.1007/978-3-319-13021-7_6

Wang, X., Sun, H.L., Deng, T., *et al*., 2015. On the tradeoff of availability and consistency for quorum systems in data center networks. *Comput. Networks*, **76**:191-206. https://doi.org/10.1016/j.comnet.2014.11.006

Wenk, A., Slater, N., 2014. Introduction. https://cwiki.apache.org/confluence/display/COUCHDB/Introduction [Accessed on May 5, 2015].

Xiao, Z.F., Liu, Y.M., 2011. Remote sensing image database based on NOSQL database. 19th Int. Conf. on Geoinformatics, p.1-5. https://doi.org/10.1109/GeoInformatics.2011.5980724

Zhang, X.X., Xu, F., 2013. Survey of research on big data storage. 12th Int. Symp. on Distributed Computing and Applications to Business, Engineering and Science, p.76-80. https://doi.org/10.1109/DCABES.2013.21

Zhao, W.Z., Ma, H.F., He, Q., 2009. Parallel *K*-means clustering based on MapReduce. IEEE Int. Conf. on Cloud Computing, p.674-679. https://doi.org/10.1007/978-3-642-10665-1_71

Zicari, R.V., 2015. On Graph Databases. Interview with Emil Eifrem. http://www.odbms.org/blog/2015/05/on-graph-databases-interview-with-emil-eifrem/ [Accessed on May 5, 2015].