Knowledge and Information Systems (2019) 60:1165–1245
https://doi.org/10.1007/s10115-018-1248-0

SURVEY PAPER

The big data system, components, tools, and technologies: a survey

T. Ramalingeswara Rao[1] · Pabitra Mitra[2] · Ravindara Bhatt[3] · A. Goswami[1]

Received: 7 December 2017 / Revised: 24 June 2018 / Accepted: 4 August 2018 /
Published online: 18 September 2018

Abstract
The traditional databases are not capable of handling unstructured data and high volumes of real-time datasets. Diverse datasets are unstructured lead to big data, and it is laborious to store, manage, process, analyze, visualize, and extract the useful insights from these datasets using traditional database approaches. However, many technical aspects exist in refining large heterogeneous datasets in the trend of big data. This paper aims to present a generalized view of complete big data system which includes several stages and key components of each stage in processing the big data. In particular, we compare and contrast various distributed file systems and MapReduce-supported NoSQL databases concerning certain parameters in data management process. Further, we present distinct distributed/cloud-based machine learning (ML) tools that play a key role to design, develop and deploy data models. The paper investigates case studies on distributed ML tools such as Mahout, Spark MLlib, and FlinkML. Further, we classify analytics based on the type of data, domain, and application. We distinguish various visualization tools pertaining three parameters: functionality, analysis capabilities, and supported development environment. Furthermore, we systematically investigate big data tools and technologies (Hadoop 3.0, Spark 2.3) including distributed/cloud-based stream processing tools in a comparative approach. Moreover, we discuss functionalities of several SQL Query tools on Hadoop based on 10 parameters. Finally, We present some critical points relevant to research directions and opportunities according to the current trend of big data. Investigating infrastructure tools for big data with recent developments provides a better understanding that how different tools and technologies apply to solve real-life applications.

Keywords Big data · Components of big data system · Distributed file systems · NoSQL databases · Visualization · SQL Query tools · Data analytics

1 Introduction

In present technological days, everything depends on a large number of online transactions. The data is increasing at a rapid rate on World Wide Web. According to *worldwidewebsize.com*, the indexed web consists of at least 45 Billion web pages as of April 30, 2018 [1].

✉ T. Ramalingeswara Rao
trrao@iitkgp.ac.in ; ramthottempudi@gmail.com

Extended author information available on the last page of the article

Square Kilometer Array (SKA) is an international project with the fundamental objective of the evolution of the Universe. SKA, a collection of several kinds of antennas, called an array which extends over long distances is expected to produce 22×10^9 TB (Terabyte) of data by 2020 [2]. As the data is growing continuously from diverse, heterogeneous sources consisting of semi-structured, unstructured, and high-dimensional formats, the analysis of such large volumes of datasets has gained much importance for upcoming research projects in various disciplines. The National Aeronautics and Space Administration (NASA) is an independent agency with the objective of the Aerospace research [3]. The purpose of NASA Jet Propulsion Laboratory (JPL) is to study and investigate the robotics in the solar system in the USA. The NASA JPL receive hundreds of Terabytes of data per every hour from space missions [4]. The network of Internet Of Things (IoT) consists of millions of devices that generate a huge amount of data through vehicles positions tracking, Global Positioning System (GPS) data, etc. According to National Intelligence Council (NIC), by 2025 IoT network has many internet nodes in our daily life including Food technology, Educational Universities, Furniture, Paper documents, Healthcare, Smart city technology, Transportation & logistics, [5,6]. Further, due to the increase in the number of online social networks, transactions on World Wide Web, the storage and analysis of large unstructured and heterogeneous datasets for extracting meaningful insights is a complex task. The process of analyzing the large-scale unstructured datasets has led to a new term known as "big data". The goal is to present how the big data can be processed to extract meaningful insights using various technical aspects. The knowledge of big data techniques is essential to spread across various disciplines to develop and discover new things that are beneficial to a variety of applications. The analysis of the diverse and heterogeneous combination of large datasets yield new information which is useful to develop new statistical models, techniques, and methods for predicting the user behavior in various applications of several disciplines. Inventing new techniques for the refinement of large datasets are helpful in taking faster decisions. The operating efficiency of data management and performance of predictive models using new techniques can be improved. The emerging research areas according to the current trend of big data are *Networking for big data* [7], *Multimedia big data* [8] and *Security issues in IOT* [9]. Complex networks from different domains (Biological/Social networks, Professional networks, etc.) generate huge volume of network data. The Multimedia big data generate data from hot daily news, events happening across the world, special show events, and predicts user's behavior and priorities. Security issues arise in the areas relevant to an application, infrastructure and communication in IOT to address threats/vulnerabilities of IOT environment. The objective is to provide a comprehensive overview of the big data system in a generalized perspective that involve several stages and to explore various components in each stage to process the large-scale datasets. So that a reader can have a better understanding of processing massive datasets that lead to identifying new problems in processing complex and heterogeneous datasets. The main stages of processing large-scale datasets are data sources, data management, computing frameworks, and data analysis. The purpose is to emphasize the whole big data system by illustrating various components with the comparison of available advanced tools along with their recent developments, strengths, and shortcomings. Big data analytics for decision making has the potential for effective risk management, faster business growth, better Return-on-Investment (RoI), and an increased level of customer satisfaction.

1.1 Big data

Existing literature investigates various definitions of big data. Large-scale datasets are the combination of large, complex, diverse, and heterogeneous datasets which generate from various sources such as clickstreams, sensors, sharing videos, business transactions and social networks. A comparative approach to define big data is, large-scale datasets which are beyond the capabilities of generally used conventional data management and analytical methods to capture, store, access, manage, share, process, analyze, and visualize within an acceptable elapsed time [10–13]. The word "big data" has accelerated to a new revolution in the data management process of large-scale big data in various disciplines of science and technology. Hence, innovative tools and techniques have to develop in data analytics to meet the challenges and targets in analyzing large-scale datasets. Therefore, we require a framework which maintains high computational speed, capable and scalable storage systems for large-scale datasets. The generation of large-scale data causes three major challenges. They are data volume, data velocity and more variety forms of data [14]. These properties are referred as *3V*-model (volume, velocity, variety) for big data. Further, the model extends to *5V* model for defining the characteristics of data that consists of volume, velocity, variety, veracity and value [15–23].

- *Volume* is the magnitude of large-scale datasets. The variation in the size of large-scale data relies on the structure and time of data, i.e., volume is the size of distinct types of data acquired from distinct data sources. Social media data (Facebook, LinkedIn, Twitter, etc.) is huge in volume. The total volume of the data across the globe by 2020 will be more than 40 Zetta Bytes from the reports given by International Data Corporation (IDC) [24].
- *Velocity* is the rate at which the data is received and then refined for analytical purposes. The speed of the data relies on the concept of data streams which work on a sliding window buffering model [25]. Social networks like Facebook and Youtube generate data streams continuously through photographs and videos sharing.
- *Variety* is a distinct type of data representation such as structured, semi-structured, and unstructured data. Some examples of structured data are relational data, tabular data and Electronic Health Record (EHR) data. Data from smartphones, log data, audio, video, image, blogs and network data is an example of unstructured data. HTML and XML are relevant to semi-structured data type.
- *Veracity* refers to the biases, noise, and abnormality in data. Veracity issues arise due to the process uncertainty (randomness in process), data uncertainty (data input uncertainty), and model uncertainty (approximate model) [26]. Users opinions in social networks about particular incidents are uncertain, because users have their own opinions which are not unique. Still, the data contains precious information. Hence, handling the uncertainty of data is another aspect of big data that can be managed by big data tools and analytics [27].
- *Value* is another dimension of big data in business perspective. Business organizations need to notice the value of big data, to increase the profit by minimizing the operational costs to provide better services to the customers [12,28]. The data in their original form has less value, but applying data analytics will change the data into a high-value asset.
- *Variability* represents the dynamic changes in the data flow rates [27]. The rapid growth of data velocity from a wide range of data sources causes data flow to fluctuate in unpredictable ways, and managing variations of data is a challenging task with a limited number of computing facilities.

– *Visualization* The author Sun-Yuan Kung added another feature to "V"-Paradigm of big data, known as Visualization [29]. Visualizing the intuitive insights of hidden structures of the data is helpful in taking strategic decisions for creating new business value [30].

1.2 HACE theoerem

– Wu et al. [31] proposed *Hace* theorem, which states that the investigation of big data begins with huge volume, heterogeneous data from autonomous data sources, aiming at complex and evolving relationships among data. The HACE theorem combines *4P model* (Predictive, Preventive, Personalized and Participatory demands), 4V model (Volume, Velocity, Variety and Veracity), and *Five R Model* (Relevant, Real-time, Realistic, Reliable and ROI) to form a framework for refining massive datasets [32].
– *BigKE (Knowledge Engineering with big data)* is a fully funded project by Ministry of Science and Technology, China, for top 15 research and development organizations to investigate the basic theory and distinct applications of big data. BigKE [32,33] is a framework that consists of a three-tier approach. The first tier is *fragmented knowledge modeling and online learning* generated from data sources, which manages the uncertainty, heterogeneity while producing fragmented knowledge. The second tier is a *nonlinear fusion of fragmented knowledge*, which synthesizes the fragmented knowledge to obtain integrated knowledge using graphical knowledge representation strategy, appropriate for knowledge fusion and navigation. Finally, the third tier of BigKE is *automated demand-driven knowledge navigation*, which involves social and personalized modeling in association with context perception. The method of connecting fragmented knowledge for personalized requirements in a knowledge graph is known as navigation. The BigKE project serves the people according to their needs in various domains like health care, education, agriculture, power sector, and real-time governance. For example, in an education sector, when a child is going to complete his graduation in the USA, and the parent in another country wants to send him for post-graduate courses in a different country. The parent decides the best university for his child by knowing the best universities, course discipline, economic fee structure, laboratory infrastructure facilities, accommodation, and job placement details provided by the BigKE project.

In a broader view, big data is not only concerned with distinct traits of large-scale data, but big data has a vast potential in various branches of Science and Technology such as Cancer Research, High Energy Physics, Genomics, Bioinformatics, Highway & Transportation Engineering and Geospatial engineering. However, there is a necessity to design, discover, develop, integrate and update the new infrastructures, architectures, methods supported by tools and techniques to process the heterogeneous datasets across various disciplines.

1.3 Types of big data models

The big data models can be categorized into four types. They *Massively Parallel Processing (MPP) systems*, *MapReduce (MR)-based systems*, *Bulk Synchronous Parallel (BSP) systems* and *in-memory models* [34]. Figure 1 shows distinct types of big data models with respective frameworks. In MPP systems, data is divided into partitions among multiple servers with each server having memory to process data locally. There is no disk level sharing in MPP systems [35]. However, the substantial amount of hardware and software is the disadvantage for commercial purposes in MPP systems. TERADATA [36], Apache HAWQ [37], Greenplum

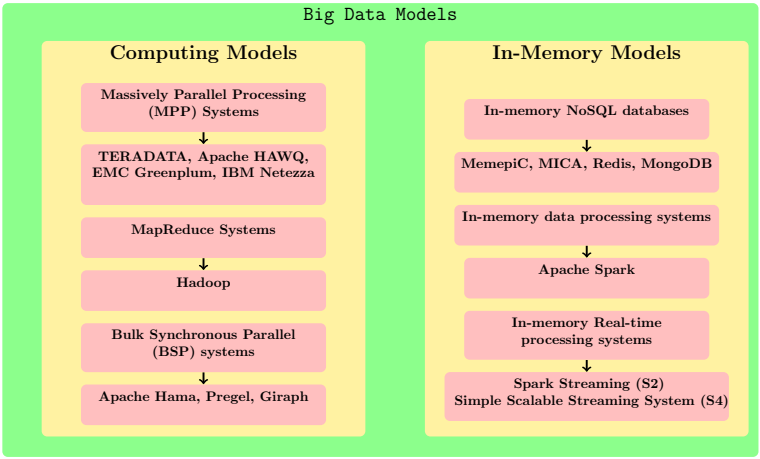

Fig. 1 Various types of big data models

[38], IBM Netezza [39], etc., are the examples for MPP-based systems. *MapReduce* is a computing framework that depends on *Map* and *Reduce* tasks [40]. Apache Hadoop is an open-source project and works on the MapReduce approach. In *BSP systems* computation takes place in iterations (supersteps). In each *superstep*, concurrent computations are to be performed in parallel on cluster nodes by using a global synchronization for information exchange between processes [41,42]. Apache Hama [43], Pregel [44] and Giraph [45] are BSP-based frameworks. *In-Memory models* are the systems in which the data storage alters from disks to main memory that leads to 100 times faster regarding response time and throughput when compared to disk-based systems [46].

The *in-memory models* can be categorized into *in-memory database systems*, *in-memory data processing systems*, and *in-memory real-time processing systems*. *In-memory database systems* are similar to MPP systems, but in execution, each server has a limited quantity of memory. MemepiC [47], MICA [48], Redis [49], and MongoDB [50] are some of the in-memory database systems which we discuss in Sect. 3.2.3. *Apache Spark* is an *in-memory processing framework* which works on the notion of *Resilient distributed datasets*. We present detailed technical aspects of Spark including recent developments in Sect. 4. *Spark Streaming*(S2) and *S4* are the *in-memory Real-time stream processing systems* that we explain in Sect. 4.4.

1.4 Our contributions

- We describe a theoretical view of the generalized big data system that illustrates that how to store, load, process, analyze, and visualize massive datasets. We present several stages that exist in big data system
- We emphasize the potential aspects of key components in each stage. We compare and contrast several distributed file system in respect of scalability, operations, storage management, replication, and throughput and latency parameters and draw important observations. Also, we compare and contrast several MapReduce-supported NoSQL databases concerning ten parameters.

– We distinguish various distributed/cloud-based machine learning (ML) tools and demonstrate various case studies on how ML tools are used to analyze massive datasets.
– We classify Analytics for big data based on data type, domain, and type of application. Moreover, We focus on different visualization tools with their strengths and shortcomings.
– We present big data technologies and tools with recent developments. We investigate Hadoop 3.0, SQL query tools on Hadoop, Spark 2.3 and Flink 1.4. Also, we compare several stream processing systems and highlight the advantages and limitations of each pair of streaming tools. Moreover, we discuss cloud-based streaming solutions for handling massive datasets.
– Furthermore, we emphasize Interactive analytical processing tools (Hive, Impala and Tez), data ingestion tools (Flume, Sqoop, Chukwa). Finally, we present the core features of recently developed large-scale graph processing tools: GraphX and Gelly.

The knowledge of generalized big data system with several tools and technologies with recent developments provide a better understanding of how massive datasets can analyze and apply to solve real-world applications.

2 Related work

The traditional approach of handling the structured data comprises of two steps. First one is storage schema, and the second step is relational database management systems for data retrieval [51]. Mazon et al. [52] argued that the OLAP tools fail for dealing with complex and heterogeneous data in real-world applications, because they only allow homogeneous dimension hierarchies. The traditional databases like RDBMS are well suitable for structured data and inefficient in handling unstructured datasets, and they do not parallelize well for heterogeneity of data [11,17,53]. Data is arriving from distinct sources with complex and evolving relationships among data in diverse dimensions causes to "big data" and keep growing with huge volumes in size (Petabytes, Exabytes, Zettabytes) [31]. Gandomi et al. [27] highlighted that the size of the data is only one feature of big data, variety and velocity are also other influential characteristics of big data. Initially, the word big data was named after *3Vs* − volume, velocity, and variety of data [10,14,22], and later *4th V* added which is meant for value [51,54]. Fernandez et al. [17] described in their work about significance of big data development in Business Intelligence, addressing big data problems, cloud environments for big data, storage system, data management, execution environment and query systems for the analysis of massive datasets. Assunccao et al. [18] defined five characteristics of large datasets as −variety, velocity, volume, veracity and value of big data. Chen et al. [51] presented different phases in the data management process, and applications of big data in several disciplines of scientific research organizations. The inconsistency of data, scalability, access, sharing, analysis, security issues, are the major challenges in big data [51, 55–57]. Kambatla et al. [58] provided different platforms for hardware (memory, processing hardware, network resources for large datasets, power consumption in loading the data), software−" storage platforms, data processing models and message passing/shared memory models", applications in several disciplines and new trends in big data analytics. Hashem et al. [12] stratified big data into five different categories. Moreover, the authors compared the characteristics of several storage media and discussed some case studies related to big data technologies. Further, a study on SQL tools on Hadoop was presented by [59]. Zhang et al. [46] explored in their study on various in-memory technologies for big data processing including core memory features, in-memory relational/NoSQL databases, in-memory cache systems,

and in-memory real-time processing systems. Kune et al. [22] investigated the differences between conventional data warehousing and big data processing concerning volume, speed, data representation, query languages, and cloud computing platforms. Big data becomes useful if one can analyze the data using analytics in an appropriate manner. Analytics is a multidisciplinary field that uses predictive modeling, statistical and data mining techniques to analyze the large-scale datasets to extract hidden insights. Analytics are of three types: descriptive, predictive, and prescriptive [60]. Yang et al. [61] explored in their review, the effects of cloud computing to handle large-scale datasets in the digital earth pertinent to several fields of Science and Technology. Ahmed et al. [62] presented in their survey about the tools that are crucial in different layers of big data processing including Hadoop distributed platforms.

Table 1 illustrates some of the recent research papers on big data and related technologies. All the existing surveys [17,22–24,46,51,53,62–64] focuses on fundamental concepts, different layers of big data with several processing tools such as storage processing tools (GFS, HDFS), computing frameworks (Hadoop, Spark), stream processing tools, and other developed early stage tools such as HCatalog, Hive and PIG. Unlike other survey papers, we elaborate an in-depth survey with recent technological enhancements (Hadoop 3.0, Spark 2.3) in a generalized perspective of big data system by systematically investigating all aspects of different stages while analyzing the large-scale datasets. We now provide the key stages and components that involve in processing the big data in the next section.

3 Components of big data system

In this section, we present the systematic stages in processing the massive datasets. Figure 2 illustrates the key stages and components to store, manage, process, analyze, and visualize the large-scale datasets. The major stages in refining the big data are i) Data capture ii) Data management and iii) Data analysis [18]. We now discuss each of the components in each stage in the following subsections.

3.1 Data capture

The primary tasks are the generation of data and capture the data. The data can be captured from heterogeneous sources such as server logs, sensors, and search engines. The majority of data sources are user-generated data and machine-generated data that form the massive datasets.

3.1.1 Data sources

The three different types of data are i) structured (electronic health record data, relational data), ii) semi-structured (user tweets, log files), and iii) unstructured data (unstructured clinical data, protein–protein data, social network data, videos, images, audio recordings).

- *User-generated data* is the data generated by users on a daily basis and their contribution of data that leads to useful information to others [65]. Tweets on twitter, movie ratings, blog information, and mobile data are the examples of user-generated data.
- *Machine-generated data* consists of the combination of all activities in the form of records comprises of users behavior, customers transactions, server activities and log data [66].

Table 1 Some valuable research articles on big data and related technologies

Author	Overview	Content	Technologies/applications
[63]	Apache Spark	Overview, high-level architecture of Spark	Spark packages, MLlib, Stream processing, GraphX
[24]	big data 2.0	Large-scale data processing platforms	Hadoop, Spark, Flink Pregel family, GraphLab, GraphX
[23]	Social big data challenges	Methodologies for Social big data	Applications in Marketing, Crime analysis, Epidemic Intelligence, Data Visualization methods, privacy issues
[11]	big data survey	Challenges, Relationship between big data and IoT, Storage mechanism and architecture for big data	Big data enterprises, IoT-based big data, Smartgrid, Health and medical care
[22]	Big data taxonomy	Layered architecture of big data clouds	Big data platform services
[12]	Cloud computing and big data	Classification of big data, storage system, comparison of various big data cloud platforms	Privacy preservation in clouds MapReduce related projects and various SQL interfaces
[17]	big data with Cloud computing	Big data framework Query systems for big data	MapReduce, NoSQL databases, SQL query tools, Machine learning algorithms
[53]	Scalable systems for big data analytics	Distinction between big data and traditional data. Layered architecture of big data	Programming models, Types of big data analytics
[51]	Big data design principles, challenges and opportunities	Big data tools and technologies	Batch processing, Stream processing and interactive analysis tools
[46]	In-memory big data management and processing systems	Core technologies for In-memory systems	In-memory data storage systems, In-memory NoSQL databases In-memory cache systems In-memory Real-Time systems

Table 1 continued

Author	Overview	Content	Technologies/applications
[62]	Big data technologies	Big data and Hadoop ecosystem tools	Data Storage tools
			Data processing tools
			Data access and Management layer tools
[64]	Stream Processing systems	Core technologies for steaming systems	Online data processing architecture, Stream processing engines and Techniques for elastic stream processing
Our paper	Generalized big data System	Major components of various stages for big data processing	Distributed file systems NoSQL/In-memory NoSQL databases. MapReduce 2.0, Hadoop 3.0, Spark 2.3
			Classification of Analytics
			Visualization tools
			ML case studies distributed Streaming processing systems
			Cloud-based ML tools
			Cloud-based Streaming analytics
			Container orchestration tools distributed Messaging systems

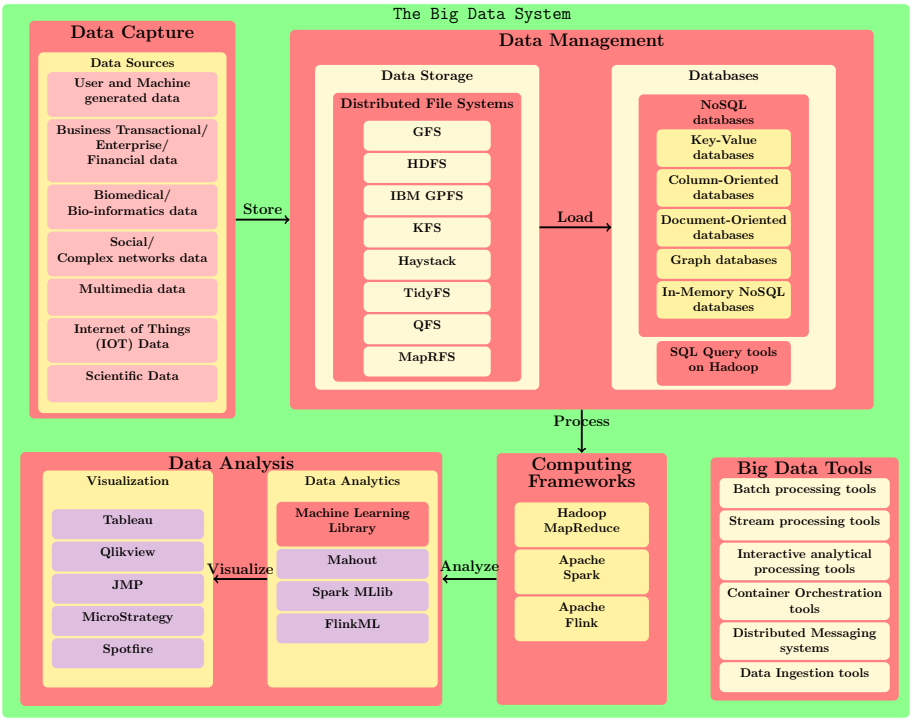

Fig. 2 An overview of big data analysis

Machine-generated data includes click stream data, configuration files, database audit logs, call center records, machine-readable object data (Barcodes) in various organizations. Most of the industries have their own machine-generated data and hence, the data comes in many unpredictable formats.

Rob Kitchin has described three kinds of big data sources in his work. They are automated, directed, and volunteered systems [79,83]. Clickstream data generated from different online Web sites that record users navigation behavior, data from sensors installed in several objects, and machine–machine conversations data over the internet are some of the data sources related to the automated systems. The directed systems rely on data generation centers operated by human beings. CCTV (closed-circuit television) cameras installed in shopping malls, business organizations, etc., generate live data which will be monitored and controlled by administrative clerks. Also, immigration data at passport office collects and inspects the passenger details against security issues by generating new images, fingerprints or iris scans. In volunteered systems, users upload their personal details in the Web sites of government organizations, users interactions in social media and crowdsourcing data where users create and upload the data to a common platform causes to different sources of big data. We stratify some of the major data sources according to volume, velocity and variety. Numerous data sources are available in various fields of science and technology. Some of the data sources are given in Table 2.

Table 2 Data Sources

Data sources	Volume	Variety	Velocity
Biomedical data/Bioinformatics data [67,68]	The storage capacity of data at EMBL-EBI (European Bioinformatics Institute) as of September 2017 is 120 Petabytes and the data was doubled from the past 2 years [69]	Protein–protein interaction (PPI) data, omics data, protein sequence data (sequences, families and motifs), gene expression data (RNA, DNA, metabolite expressions)	Real-time biomedical data stream refers to the collection of information obtained by the aggregation of data of individuals using health-tracking devices or data generated from healthcare software systems during patient visits and diagnosis of different diseases
Business transactional/Enterprise data [25]	Walmart's data warehouse consists of 2.5 Petabytes of information per hour from consumers transactions [70]	E-commerce transactions, ERP data, CRM data, Supply chain data, logistics, procurement data	Recommendation engines developed by Amazon modifies pricing across competing sites for every 15 seconds [70]
Financial data [71]	NYSE Technologies acquires five Terabytes of data per day and utilizes the data for complex query analytics, market surveillance and capacity planning [72]	Banking transactions, mobile banking data, online banking data and loan data	Financial Organizations track variations in the stock market and modifies portfolios based on stock price movements
Health-Informatics and Clinical data [67,73,74]	The total volume of healthcare data is increasing at an astronomical rate: 153 Exabytes (one Exabyte = one billion gigabytes) were yielded in 2013 and by 2020 it is expected that the healthcare data will be reached up to 2,314 Exabytes, and the overall rate of increase is at least 48 percent annually [75]	Pathology, radiology data, Epidemiology & Behavioral data, Magnetic resonance imaging (MRI) data, EEG data, Electronic Health Records (EHR) data, pharmaceutical data, health sensing data, and biometric data	Observing clinical signs of patients like skin color, the mood from real-time or recorded video feeds. Data generated using health-tracking wearable devices (distance walked, number of steps counted) heart rate variability data using biosensors

Table 2 continued

Data sources	Volume	Variety	Velocity
Social Network data [76,77]	Facebook has more than 2.13 billion monthly active users for Quarter 4, 2017, which is a 14 percent increase a Year over Year [78]. The popular social network Twitter had 328 million monthly active users by the first quarter, 2017 [76]	Facebook posts via text, images and videos. Twitter tweets audio, video, image and text messages	Facebook users are uploading 300 million photographs per day [79]. Twitter sent 330 million tweets per day, and it generates on an average 6000 tweets per second [77]
Scientific data [68,80]	Large Hadron Collider produces 15 Petabytes of data per year [68]	Astronomical data, remote sensing satellite data	Large Hadron Collider consists of 150 million sensors and producing data 600 million times per second [80]
Internet Of Things (IoT) data [81,82]	The IoT network is expecting to reach 212 billion devices deployed worldwide by 2020 [81]	Global Positioning system (GPS) signals, images, text and videos through mobile applications like WhatsApp	The IoT network generates per every millisecond massive volume of data from millions of accessories in the field of vehicle tracking, healthcare, educational institutions, agriculture, transportation and from several industries

3.2 Data management

In this section, we present one of the main contributions of the paper. Once the data is generated from different sources, the immediate crucial stage is *data management* which can be categorized into two subtasks. They are *data storage* and *data retrieval*. The big data applications work on a large-scale basis. The distributed file system handles the responsibility of large-scale data storage. The second task is *data retrieval*, which is handled by NoSQL databases. Further, MapReduce is a distributed computing framework that permits massive datasets over a set of nodes. We present the characteristics of a distributed file system and comparison among several file systems relevant to various parameters.

3.2.1 File system

The distributed file system is the main component for storing massive datasets. We compare and contrast several distributed file systems in respect of the parameters given in Table 3. A distributed system is characterized by the properties: reliability, scalability, availability and efficiency [84].

The purpose of the distributed file system is to store and share files over a set of connected nodes. Several distributed file systems are Google File System (GFS), Hadoop distributed file system (HDFS), Haystack File System, Tao file system, Kosmos file system, TidyFS, IBM General Parallel File System (GPFS) and Quantcast File System (QFS). Google File System (GFS) distributes three copies of each piece of information in such a way that it can recover the data when the failure cases occur [85,86]. Shortage of POSIX (Portable Operating System Interface) features and high storage overhead are the limitations of GFS. POSIX is a set of standards described by the IEEE[1] computer society for sustaining compatibility between operating systems. However, IBM GPFS is the one which supports POSIX semantics [87]. Table 3 illustrates the comparison among several distributed file systems with respect to the parameters read/write, storage management, replication, throughput, and response time. These parameters have a significant role in measuring the performance of distributed file systems.

- *Scalability*
  All the distributed file systems are scalable and stores petabytes of data. QFS needs less space when compared to HDFS due to its storage structure. In the current version of Hadoop 3.x, the default block size is 128 MB. In IBM version 4.x, the file system limit is 18 PB and the number of disks are increased from 2048 to 4096.
- *Read/Write operations* HDFS, KFS, Haystack works from the perspective of *write once* and *read many times*. However, GFS and TidyFS support *read once* and *write once* only. Moreover, MapRFS supports for full read/write data storage and enables files to be modified.
- *Storage Management*
  GFS maintains storage of the data over multiple number (Thousands) of disks. The HDFS uses DataNode and NameNode to store the data by having the default replication factor as three. However, GPFS allows multiple computers to access a common set of file data concurrently over a Storage Area Network. Haystack maintains a custom storage system using NFS (Network file system) designs and Content delivery network. The storage system in TidyFs comprises three components: a metadata server, a node service, and

[1] Institute of Electrical and Electronics Engineers.

Table 3 Comparison of various distributed file systems

	Scalability/Storage capacity	Read/write	Storage management	Replication	(Bandwidth) (Bandwidth)	Latency/response time	Application software
GFS [85,86]	Scalable	Read once/write once	Storage across thousands of disks	Asynchronous	High aggregate throughput	Not low-latency	Google BIGTABLE
HDFS [90]	The default block size is 64 MB. The block size in Hadoop 2.0 is 128 MB based on workloads [91].	Write once Read many times read speed 50 MB/s	NameNode DataNode	Asynchronous Replication factor is 3	High	Reduces Write latency	Suitable for Batch processing jobs (MapReduce)
IBM GPFS [92,93]	Currently IBM version 4.x file system limit is 21.6 PB. No.of disks in the file system are increased from 2048 to 4096. [94]	Read/Write	Over Storage Area Network	Synchronous/ Asynchronous	Full throughput	Low	IBM Systems
Kosmos File System (KFS)[95]	Multiple Petabytes of data storage	Write-once read-many times	Decoupling storage from filesystem namespace	Asynchronous	N/A	N/A	Hadoop

Table 3 continued

	Scalability/Storage capacity	Read/write	Storage management	Replication	(Bandwidth) (Bandwidth)	Latency/response time	Application software
Haystack [96]	More than 1000 NFS file servers. over 35PB of configured capacity	Write-once read-many times	Custom storage system. NFS (Network File System) designs and content delivery network (CDN)	Replicates each photograph in geographically distinct locations	High	Low	Facebook
TidyFS [97]	Scalable	Read/write once using native interfaces. data is immutable	A metadata server a node service and TidyFs explorer. Data storage as blobs which are immutable	Lazy replication	high	low	Dryad, support MapReduce
Quantcast File System (QFS) [98] [87]	Highly scalable 50 percent less disk space than HDFS	Read speed 300 MB/s. Faster writes using Leaner data encoding.	Storage savings by employing erasure coding rather than replication	Reed–Solomon erasure coding	High	Low	Suitable for Batch processing jobs (MapReduce)
MapR File System[99] (MapRFS)	Scalable	Full read/write data storage. Allows files to be modified overwritten and read as required	Every node in the cluster stores and serves metadata to avoid loss of data and disk/node failures	Online point-in-time data recovery without costly replication of data	High	Low	Support hadoop

TidyFS Explorer (TE). TE is a graphical user interface that enables users to observe the state of the system. Rather than replication as a fault-tolerance QFS employs an error correction technique known as *Reed–Solomon encoding*, which provides a better data recovery power and requires only 50 percent data expansion. Hence, QFS needs only half the data storage than what HDFS is required for equivalent storage capacity [87]. QFS maintains high-speed rate for reading and faster writes using leaner data encoding. Each node in MapRFS in the cluster stores the metadata to overcome disk and node failures.

– *Replication*
The replication in GFS, HDFS, KFS is asynchronous, whereas IBM GPFS follows both synchronous and asynchronous replication and TidyFS has lazy replication. In the synchronous (or eager) system, the data transmission delay across a communication channel is bounded by a value, i.e., when a client gets a response from the server, the response is sent only after all necessary operations has finished. Unlike in the synchronous system, asynchronous or lazy replication has no bound for data transmission delay, i.e., instant response and propagation of changes and coordination among all replicas are done subsequently [88,89].
– *Throughput and latency*
HDFS, Haystack, TidyFs, QFS and MapRFS produce high throughput in less response time.
– *MapReduce support*
HDFS, KFS, TidyFS, QFS and MapRFs are suitable for batch processing jobs. The advantage of TidyFs compared to GFS and HDFS is its native data access. TidyFs empowers input–output (I/O) operations with the help of acceptable access patterns and compression techniques. In HDFS, if the NameNode fails, the complete cluster is unaccessible and it may take minutes or hours to restore the NameNode. However, MapR file system has a convenience in which every node in the cluster stores and distributes metadata to avoid loss of data due to multiple disk/node failures. Instead of the write-once system like in HDFS, MapRFS permits files to be modified, overwritten and read as required. MapR snapshots provide a point-in-time recovery image for data recovery that protects from user/application errors.

Relational database management systems could not handle with unstructured data due to their limitations. They do not parallelize well for heterogeneous datasets. Non-relational (Not Only SQL-NoSQL) databases are developed in view of horizontal scalability which works in a distributed environment rather than depends on highly available hardware. We now present Hadoop-supported NoSQL databases that play a prominent role in data management of big data.

3.2.2 NoSQL databases

In traditional relational databases, all the attributes of an entity are stored in the form of rows. To retrieve some attributes from the database entire record has to be read. Since the traditional row databases were designed for transactional processing, they can handle only for smaller datasets. The major distinction between NoSQL systems and traditional databases is that NoSQL systems follow the horizontal scalability of the data with respect to storage and retrieval of key-value pairs, whereas traditional databases depend on primary/foreign-key relationships [12]. NoSQL databases work efficiently with unstructured data. NoSQL systems follow the CAP theorem discovered by Eric Brewer that no distributed system can

satisfy *consistency*(C), *availability*(A) and *partition tolerance*(P) concurrently [100,101]. *Consistency* is the ability of a system that propagates the same and current updated form of data to all nodes and guarantees the consistency of data on replicated partitions of the database. *Availability* refers to the ability of a system that provides reliable service even when the failures (crash, unavailability) occur at the replicated partitions of the database. The failures must be identified at the earliest and rapid recovery mechanism must be initiated [84,102]. *Partition tolerance* is the capability of a system to proceed for further operations even in the case of network failures.

- *Column-oriented databases*: Rather than reading entire record, column-based systems allow reading just the required attribute [103]. In column-oriented databases, each attribute is placed in a different region on storage, and they follow vertically split-up database into a set of independent columns that are reposed individually. They need smaller disk space and allow parallel data access for rapid joins and aggregations in improving the machine performance [104]. Column families that are consistent with replication and scheduling limitations of Hadoop improve the speed of MapReduce jobs [105]. The major column-oriented NoSQL databases are BigTable [106], Cassandra [107], c-store [108], HBase, HyperTable, MonetDB [109,110], SimpleDB [111] and VectorWise [112].
- *Documented oriented databases*: These databases consist of a collection of documents as key-value pairs plus a payload represented in the form of JSON, XML, and Binary formats. These databases allow embedding of documents and maintain horizontal scalability to achieve high performance. Document databases are not suitable for the data that can fit into tables and extremely normalized. MongoDB [113], Terrastore,[2] CouchDB,[3] RavenDB [114] and Couchbase [115] belong to the class of document databases.
- *Key-value databases*: The main purpose of a key-value store is the ability to store some data, called a value, inside a key. The stored data can be retrieved, only if we know the exact key used to store it. In these databases, the records consist of key-value pairs that are implemented using distributed hash tables. Maintaining horizontal scalability, simplicity, and high performance are the advantages of key-value databases. Examples for key-value databases are Dynamo [116], Redis [49], Riak [117] and Voldemort [118,119].
- *Graph databases*: These databases stores the data represented in terms of nodes or objects connected by some relationships. Unlike other databases, the data structure in Graph databases comprises mainly three characteristics known as nodes, edges and properties [102]. AllegroGraph [120], HyperGraphDB [121], Infinite Graph [122], InfoGrid [4] and Neo4j [5] are some of the graph databases [123].

The major MapReduce-supported NoSQL databases are Riak, Voldemort, BigTable, HBase, Cassandra, HyperTable, Accumulo, MongoDB, CouchDB, RavenDB and Couchbase. They are compared with respect to the parameters scalability, replication, operations, data encryption and limitations (See Table 4). These parameters perform a vital role in managing the large-scale datasets in big data analysis. All these databases satisfy the characteristics scalability and availability and support MapReduce with high performance. We now give a brief introduction about the parameters that are necessary for the data management process.

[2] https://db-engines.com/en/system/Terrastore.

[3] http://couchdb.apache.org/.

[4] http://infogrid.org/trac/.

[5] https://neo4j.com/.

- *Latency* is the measurement of the time between a client request and the actual server response.
- *Scalability* refers to the ability of a system to incessant progress and to handle increasing bulk volume of data without loss of performance [84]. Horizontal scalability can be attained by adding new host machines. However, the vertical scalability can be maintained by the addition of multiple resources to a single host machine.
- *Replication* is the ability to achieve fault-tolerance by sharing the same data among database server nodes, so that even if few nodes go down, all the data should be retained.
- *Availability* refers to the ability of a system to restraint as good as achievable the latency [84]. The failure (crash/unavailability) must be recognized as early as possible and a fast recovery action should be performed. The mechanism for maintaining a reliable system to identify and fix node failures rapidly is called failover.
- *Sharding* is a method to distribute data over multiple nodes.

Table 4 illustrates various MapReduce-supported NoSQL databases, and we conclude the following key points.

- In *key-value databases*, Riak does not support atomic operations and provide eventual consistency. In eventual consistency, the storage system gives assurance that if no new changes are applied to an object, eventually all accesses will go back to the earlier modified value [129]. Riak can accumulate only a confined number of keys per node. Both Riak and Voldemort do not support data encryption. Voldemort supports per-tuple-based replication in which each data item is automatically replicated over multiple servers.
- In *Column-oriented databases*, HBase does not support data store encryption and protocol encryption. However, HBase support server-side encryption as HBASE-7544 to overcome the sudden leakage of data at rest [130]. The limitation of HBase is the cell value size which is limited to 20 MB. Cassandra supports client-to-node and transparent data encryption. In client-to-node encryption, the data in flight from client machines to a database will be protected by using secure socket layer. Transparent data encryption enables to encrypt data at rest and the data cannot be readable by unauthorized users. The column database HyperTable has no data storage encryption facility. Also, there is a loss of data in HyperTable due to frequent failures in the master node. Cassandra does not support join operation and aggregate functions, whereas HBase support joins and all other CRUD (Create, Read, Update and Delete) operations. Both Cassandra and HBase are good for handling all operations, but HBase has less secure facilities compared to Cassandra.
- In *document databases*, MongoDB achieves horizontal scalability through automatic sharding by distributing data across several nodes with automatic load balancing. The replication in MongoDB is master–slave, in which slaves request the master for all write operations. MongoDB ensures eventual consistency. MongoDB supports both data store encryption and protocol encryption. CouchDB supports only protocol encryption but does not support data store encryption. The drawback of CouchDB is documents cannot be nested. MongoDB, CouchDB and RavenDB work on mater–master replication. MongoDB also supports encryption for the data at rest, storage engine, application level and at third-party storage. The authenticated encryption in MongoDB Enterprise is AES256-GCM (a 256-bit Advanced Encryption Standard in Galois/Counter Mode) [131]. The limitations of MongoDB are, it does not support joins and is not suitable for transactional systems. For example, MongoDB is not applicable for banking transactions. Another limitation is by default the connections to and from MongoDB are not encrypted [113]. When CouchDB follows master–master replication, then the consistency is eventual consistency

Table 4 MapReduce-supported NoSQL databases

NoSQL databases\language	Scalability	Replication	Operations	Hadoop	Data encryption	Other features/ limitations
Riak [117] (Key-Value) Erlang	Horizontal Scalability	Asynchronous Eventual consistency	Doesn't support atomic operations.	Support MapReduce	No	Can accumulate a confined number of keys per node.
Voldemort [118] (Key-Value) Java	Scalable	Per tuple-based replication.	Adding, deleting, stores. Retrieving, deleting, updating partitions	support	No	N/A
BigTable [106] (column database) Java	Scalable	Asynchronous. Ensures eventual data consistency	Creating, deleting tables and column families, metadata modification	Support MapReduce	Encrypted both in-flight and at rest	Row keys are arbitrary strings with maximum size 64KB.
HBase [124] (column database) Java	Scalable	Master–Slave/Master–Master replication. Ensures eventual data consistency	Read, write, update and delete tables, supports joins	Support hadoop and zookeeper	Transparent server-side encryption as HBASE-7544	Cell value size is limited to 20MB
Cassandra [107,125] (column database) Java	Scalable	Replication supports within each cluster between geographically clusters. Ensures eventual data consistency	No joins, no aggregate functions, create, alter, update, delete on keyspaces, tables, and indexes using CQL	MapReduce	Support Client-to-node and transparent data encryption	Maximum size of an item in a collection is 64k or 2 Billion

Table 4 continued

NoSQL databases\language	Scalability	Replication	Operations	Hadoop	Data encryption	Other features/ limitations
HyperTable [126] (column database) C++	Scalable	The replication of Tables is maintained and partitioned across servers in respect of key ranges.	Meta operations like table creation, deletion and range assignment. Does not support joins	Support	No	Loss of data due to frequent failures of the master node, while writing large amounts of data. HyperTable version (0.9.12)
Accumulo [127] (column database) Java	Scalable	Eventual consistency. Lazy replication	N/A	Support	Support encryption	Bulk importing of files into a table configured for replication, are not replicated
MongoDB [113] (document database) C++	Achieved by distributing documents between nodes using a shared key	Asynchronous master-slave replication. Eventual consistency	Read, insert, delete, sorting, projection and iterating through query results. Does not support joins and transactional operations	Support	Support encryption at rest, supports authenticated encryption AES256-GCM	Maximum document size is 16 MB. Maximum indexes in a collection allowed are 64. Index Keys cannot be larger than 1024 bytes.

Table 4 continued

NoSQL databases\language	Scalability	Replication	Operations	Hadoop	Data encryption	Other features/ limitations
CouchDB [53,128] (JSON-document database) Erlnag	Scalable	Master–master/master–slave replication Eventual/Strong consistency	B-tree storage engine enables searches, insertions and delete operations. Does not support transactions	Support	Does not support data store encryption and supports protocol authentication	Documents can not be nested. A limit of 8 attachments per document and each attached file size should not be greater than 64 KB
RavenDB [114] (document database) .NET	Scalable	Master-master replication eventual consistency	Perform create, retrieve, update and delete opera- tions.	Support	Documents and indexes are encrypted when they are on a disk using strong encryption.	Each file is divided into pages. A page is a sequence of bytes. Its maximum size is 64KB
Couchbase [49] (document database) C++, Erlang	Horizontally scalable	Peer-to-peer based (strong consistency) and Cross-Data center Replication (XDCR) (Eventual consistency)	CRUD operations counter, remove append, prepend	Support	Supports encryption when the data is at rest.	Maximum document size for indexing is 20 MB

and it has strong consistency when it supports master-slave replication. The advantage of Couchbase [115] is that it provides two kinds of replication. They are peer-to-peer-based (intracluster) replication and Cross-Data Center (inter-cluster) Replication (XDCR). In peer-to-peer-based replication, each node in a cluster consists of *data of that node* and *replicas of other nodes* in that cluster. In this case, Couchbase has the strong consistency in each cluster. Couchbase do not have a topology, hierarchy or master-slave relationship between nodes in a cluster. In XDCR, the replication is maintained between different geographically located clusters for disaster recovery or for faster data access to obtain data closer to its users. In this case, Couchbase ensures eventual consistency. Couchbase supports encryption when the data is at rest. Couchbase has rich abilities comparatively to other document databases.

3.2.3 In-memory database systems

In this section, we review some crucial aspects of widely used *In-memory NoSQL databases*. The big data applications concerning different domains need to process and analyze massive data in less amount of time. In-memory systems are suitable to analyze the huge amount of data within less time to overcome the obstacles of accessing and processing large-scale datasets using traditional databases. In-memory databases save the information in the RAM of big data servers and extremely minimize the storage I/O gap to provide faster response time and high throughput. We now discuss some of the crucial In-memory NoSQL database, namely MemepiC [46], MICA [48], Redis [132] and MongoDB [50].

– *MemepiC* [46,47] is a key-value based in-memory data management system that unifies data storage service and data analytics into a single system. MemepiC works on RDMA-based (Remote Direct Memory Access) message transfer protocol that combines single-sided (only requester side) and both-sided (requester and receiver side) RDMA verbs to attain high scalability. MemepiC is designed on the concept of *no-syscall* principle. MemepiC reduces the over-head causes due to system calls like privilege mode switch, additional memory copy between user space and kernel space by providing procedures to eliminate syscalls from the critical path of execution from software/hardware point of view. Hence, the data model of MemepiC comprises a low-latency storage layer which has a distributed key-value storage facility integrated with RDMA-based communication. The key-value store in MemepiC has two types of atomic primitives such as numeric/string for handling a set of arithmetic operations, and two composite operators (array/hashtable) to handle key-value pairs produce by map method in a MapReduce application. The main significant part of RDMA is the concept of *queue pair* which governs the message transmission over RDMA. A *queue pair* is a pair (send queue, receive queue) of two jobs which are meant for job invocations including send/receive requests. A send request establishes data transmission connection and receive requests, which consist of the details about storage of incoming data and their memory locations. Each of the send/receive queues associates with another feature known as *completion queue*. After completion of the job request, a job completion can be deliberately created and stored in the respective *completion queue* to alert the application of the completion of job requests. Another additional feature of MemepiC is the integration of *storage service* with *data analytics* functionality. MemepiC analyzes the data by minimizing the data movement cost, and by the synchronization of data analytics feature with the storage service using atomic operators and fork-based virtual snapshot [46]. MemepiC analytics engine has an overlapping implementation facility through which computation, data

access and communication processes can be parallelized. In order to maintain a coordination between storage service and data analytics engine, MemepiC uses an efficient synchronization mechanism known as lock-free data structures using atomic primitives (read, write, modify) and HTM (Hardware Transactional Memory). HTM provides is an alternative concurrent control mechanism for shared memory access.

– *Memory-store with Intelligent Concurrent Access* (MICA) [48] is an In-memory key/value store that attains stable high-throughput and low response time for read/write in-depth workloads for both constant and skewed key popularity. The core characteristics of MICA are *parallel data access*, *network stack* and *lossy/lossless* data structures. MICA adopts parallel data access based on the concept of *Exclusive access*, in which only a unique core can ingress a chunk of the data. i.e., there is a one-one correspondence (parallel) between cores and partitions. Each core exclusively ingresses the respective partition in parallel without inter-core correspondence. MICA uses keyhash-based partitioning, which maps keys to partitions and reduces the imbalance of distribution of requests. The operation modes in MICA are *Exclusive Read Exclusive Write* (EREW) and *Concurrent Read Exclusive Write* (CREW). In EREW mode a unique CPU core will be allocated to each partition. In this mode, no concurrent access to partitions will be allowed. The CREW mode permits each and every core to read partitions, but only a unique core can write. MICA merges the advantage of concurrent read and Exclusive write. When the load is skewed, CREW supervise the read-write contrasts efficiently. The core features of *network stack* are *network I/O* and *request direction*. The role of network I/O is to transmit data between NICs and server software. The job of *request direction* is to assign requests to the suitable CPU core for handling key/value efficiently. MICA sends the requests of clients to particular CPU cores at the NIC level utilizing the information provided by the client. MICA adopts a light-weight network stack that bypasses the kernel to reduce the overhead. MICA eliminates the overhead generated by network stack and sends the data packets to independent cores. MICA governs remote key/value requests to suitable cores to process the requests efficiently. Also, the network stack in MICA attains no-copy packet I/O and request processing. Moreover, MICA uses the data structures, *circular logs*, *lossy concurrent hash indexes (lch indexes)* and *bulk chaining* to maintain the high-speed read/write workloads at low overhead. Each partition in MICA, comprises of a unique circular log and lossy concurrent hash index. *Circular logs* govern the memory for key/value pairs in cache mode. A *hash index* constitutes several buckets, where every bucket is associated with a fixed number of index entries. Each index entry consists of the information about an item: (tag, item offset). A tag is a segment of the keyhash of an indexed item which filters unmatched lookup keys by the comparison of stored tag with the tag consists in lookup keyhash. When a particular bucket is full in the hash index, it avoids an index entry to accommodate a new item (indexing new key/value). Hence, the hash index is lossy. MICA transforms the lossy hash index into lossless by using the concept of Bulk chaining. It is alike to the concept of chaining hash tables in conventional methods and adds additional memory to the buckets that can accommodate more number of elements. When the bucket is overflow, MICA adds an extra bucket to the main bucket to form a chain of buckets. MICA refuses new items when there are no extra buckets available and sends an out-of-space error report to the client.
– *Redis* is another important In-memory key-value store, which works as a database, cache and a message broker [46,132]. Redis supports data structures (strings, hashes, lists, sets, sorted sets, hyperloglogs) with common database indexes such as bitmaps and geospatial indexes. The significant features in Redis are in-built replication, Lua scripting and

LRU (Least Recently Used) eviction, publish/subscribe messaging, scripting and transactions. Redis uses a *master-slave* replication environment, which enables Redis slave nodes (servers) to be exact duplicates of master nodes. The default replication in Redis is asynchronous with high latency and high performance. Redis slaves acknowledge the processed data details asynchronously with the master. Redis is frequently used for caching in an environment, where a fixed amount of maximum memory is allocated [133]. Redis embeds with LRU cache. When the new data enter, we need to remove or evict the old data to assign some space. Removing the data that is useful in future is a poor approach. On the other hand, evicting the data that is not useful for future requests is a good approach. The efficiency of Redis as a cache depends on how good decision it will take about what data has to evict. Every cache can serve some percentage of read queries (hits/misses ratio) ratio. A cache can preserve the keys with maximum probability of being accessed in the future. Similarly, the key with the least probability of being accessed in the future should be eliminated from the dataset. Redis is not able to select the suitable candidate for eviction. Redis runs an approximation of the LRU algorithm, by considering a sample of small number of keys, and evicts the key with oldest access time among the sample of keys. The significant feature of LRU is, we can set the precision for the algorithm by modifying the number of samples to test for each eviction. The precision set can be governed by the configuration directive *"maxmemory-samples number"* (eg. maxmemory-samples 15). The LRU algorithm in Redis is an approximation algorithm which is costly due to its more memory consumption.

Redis 4.0 [134] consists of new improved features such as Modules API, Improved Replication (PSYNC2) engine, new caching policy (LFU), asynchronous deletion operations and improvement in memory management. *Redis 4.0* provides a new caching improvement feature known as LFU (Least Frequently Used) mode which works in an efficient manner by yielding a better hits/misses ratio. Redis LFU algorithm finds the access frequency of different items and evicts the item that is least frequently used. On the other hand, the items which use frequently have more chances to remain in the memory. Redis provides two main policies to configure the LFU mode. They are *volatile-lfu* and *allkeys-lfu*. The policy *Volatile-lfu* expels the least frequently used key among the ones with an expire set. The policy *allkeys-lfu* evicts the least frequently used key from the set of all keys. *Redis cluster 4.0* [135] is a distributed implementation of Redis to achieve high performance, availability and to maintain safety writes from clients. Redis cluster is linearly scalable up to 1000 nodes with no proxies and uses asynchronous replication. Redis cluster performs all the single key commands which are available in the basic version of Redis and complex multi-key operations like Union/Intersections, where all the keys reside in the same node. Redis cluster executes a notion called *hash tags* to store some particular keys in the same node. Hash tags guarantee that the allocation of multiple keys in the same hash slot. However, single key operations are accessible at any time, while the multi-key operations may not be accessible for some time during manual reshardings. The cluster nodes in Redis are connected through a TCP bus-based binary protocol, known as the *Redis Cluster Bus*. The *Cluster Bus* governs the propagation of publish/subscribe messages over the cluster and handles the manual failovers. The network topology of Redis Cluster is a full mesh, where cluster nodes use a gossip protocol to prevent the exchange of too many messages between nodes, so that the exchange of messages should not be exponential. The limitations of Redis are as follows: a) The client has to perform sharding, in the sense that client has to recognize the network topology for distributing the keys between nodes, b) another limitation is merge operations on values are not allowed in Redis. Because values in Redis consist of lists and sorted sets

with millions of items. Since data types involved in values are semantically complex, transferring and merging various kinds of values are inadequate in terms of memory, and c) unlike the Redis Standalone mode, the Redis Cluster does not support multiple databases.

– *MongoDB* [50] is a document-oriented database with in-memory storage engine which is suitable to handle operational and real-time analytics. The responsibility of the storage engine is to manage storage of data, both in memory and on disk. MongoDB has multiple storage engines to handle different tasks. The current version *MongoDB 3.2.6* supports *MMAP (Memory Mapped)*, *WiredTiger* and *In-memory* storage engines. Initially, *MMAP* is the default storage engine, but after version 3.2 *WiredTiger* is the standard storage engine in MongoDB. *MMAP* uses memory-mapped files to store all its data. MongoDB uses memory-mapped files to manage and interact with data using the system call mmap(). The operating system moves the data from memory-mapped file into memory using mmap(). The mmap() maps the file to a location/block of virtual memory. The limitation in MMAP is, multiple writes to a particular collection have to send in a batch-wise manner, and the reads have to remain in the queue until the completion of writes [136]. Hence, MongoDB (version 3.2 onward) equipped with *WiredTiger* as a standard engine and uses the concept of *MultiVersion Concurrency Control* (MVCC) at document-level, so that multiple writes can be made for distinct documents of a collection at the same time. Instead of storing data on-disk, it preserves documents data in-memory to obtain more predictable data latencies. The in-memory storage engine achieves high throughput and predictable latency required by various data-intensive applications. The recent version, *MongoDB 3.2.6* consists of In-memory storage engine which ensures more predictable latency of database operations by avoiding disk I/O, configuration data, indexes and user credentials [50]. The In-memory storage engine operates document-level concurrency control for write operations. Hence, many clients can make changes to the collection of documents for write operations. The in-memory storage engine in MongoDB is non-persistent and does not write data to persistent storage.

Summary

This section explores the concepts of distributed file systems and NoSQL databases relevant to data management process. The major distributed file systems are GFS, HDFS, IBM GPFS, Haystack, TidyFS, QFS and MapR file systems. We observe that QFS consumes less storage space than HDFS. It requires only half storage capacity than HDFS storage volume. Also, the read speed (faster writes) of QFS is very high compared to HDFS. Haystack is suitable for storing huge volumes of images. MapR file system performance is superior to HDFS due to efficient redirection–write operations than copy-on-write operations and its data recovery works on point-in-time recovery image.

NoSQL databases follow the horizontal scalability and provide high performance in storing and retrieving the data. We differentiate the MapReduce-supported NoSQL databases based on the parameters scalability, replication, Hadoop support, encryption and limitations. Among MapReduce-supported databases Riak, Voldemort are key-value databases, BigTable, HBase, HyperTable, Accumulo are column-oriented databases, and MongoDB, CouchDB, RavenDB and Couchbase are the document-oriented databases. The limitations of all the MapReduce-supported databases are given in Table 4. Another important class in the data management process is in-memory NoSQL database category. MemepiC, MICA, Redis and MongoDB are the in-memory databases. Moreover, the core characteristics of MemepiC (RDMA queue

pair), MICA (network stack, bulk chaining), Redis (lfu, version 4.0 features) and MongoDB (MMAP, WiredTiger, version 3.2.6) are demonstrated.

3.3 Data analysis

The final stage of processing the big data is *Data Analysis*. *Data Analytics* and *Data Visualization* are the two major subtasks of data analysis. In analytics, the data is modeled by using ML algorithms and analyzed using statistical/data mining techniques to obtain hidden patterns from the data. Mahout [137] and MLlib [138] are the two popular distributed ML libraries built on top of Apache Hadoop and Spark respectively. Apache Flink is a distributed processing framework for the analysis of large-scale datasets that consists of another ML library known as FlinkML [139]. Using ML algorithms Mahout extracts useful patterns from massive datasets stored in HDFS. The ML algorithms for Mahout, Apache Spark MLlib, FlinkML and cloud-based ML tools are present in the following subsections.

3.3.1 Distributed machine learning tools

Mahout

– The aim of *Apache Mahout* [137] is to design and develop scalable ML algorithms. Mahout 0.13.0 includes *Mahout-Samsara environment*, a variety of algorithms in Scala for Spark,[6] Apache Flink[7] and H2O.[8] Samsara environment includes linear algebra operations, sparse matrix, vector computations that are compatible with MLlib of Spark. Samsara environment comprises stochastic singular value decomposition (SSVD), stochastic principal component analysis (SPCA), distributed Cholesky QR, distributed regularized alternating least squares (DALS), item and row similarity in collaborative filtering, and Naive Bayes classification algorithms. However, the earlier version of Mahout consists of various usual learning algorithms in classification, clustering, topic models, collaborative filtering, dimensionality reduction, and association rule mining.

Spark MLlib

– Apache Spark consists of a library of ML algorithms known as MLlib. The key packages in MLlib are *spark.mllib* and *spark.ml*. *Spark.mllib* is built on top of RDDs, whereas *spark.ml* is built on top of DataFrames [140]. The DataFrame API allows rich functional integration among applications of spark by performing various relational operations on RDDs [63]. MLlib supports to write the applications in Java, Python, Scala or R. *Spark.ml* consists of pipelines API for constructing, debugging and accommodating pipelines, whereas *spark.mllib* comprises several ML algorithms and utilities. Following are the currently available algorithms in Apache Spark [141,142].
– The improvements in Apache Spark 2.2 are ALS algorithm for top-k recommendations for all users, LinearSVC for the classification of data using support vector machine, and currently Logistic regression supports constraints on the coefficients while training the data [143]. The goal of ALS algorithm is to estimate the rating matrix into the product of two lower-rank matrices. In the ALS (version 2.0) method, the default values of *regParam* do not match within the same file. Because the parameter value in place set it to 1.0 and

[6] https://spark.apache.org/.

[7] https://flink.apache.org/.

[8] https://www.h2o.ai/.

in the other place it is set to 0.1. But in the Spark version 2.2, in order to match regular ALS API's default *regParam* setting the method *DeveloperApi ALS.train()* set the default *regParam* value as 0.1 instead of 1.0 [144]. Hence, the ALS method will not affect the users, as it is marked with *DeveloperApi*. In the current version *Spark 2.3*, the MLlib has the following new features and improvements to existing algorithms.

- In *Spark 2.3*, for operating MLlib models and pipelines to generate output results, now onwards all the existing fitted models and pipelines work with Structured streaming jobs (refer Spark streaming in Sect. 4.4).
- *Breaking API Changes* The class and characteristic hierarchy of logistic regression model summaries was altered to be cleaner and superior assistance for the addition of the multi-class summary. This property is a breaking change for user code that flings *LogisticRegressionTrainingSummary (LRTS)* to a *BinaryLogisticRegressionTrainingSummary (BLRTS)* [138]. Logistic Regression is a well accepted and particular case of generalized linear models, which estimates the probability of an outcome. In spark.ml, logistic regression can be applied to predict a binary out-turn by employing binomial logistic regression, or it can be applied to predict a multi-class out-turn by employing multinomial regression. *LRTS* describes an overview of *LogisticRegressionModel*. In the case of binary classification, an additional measure like ROC curve is available. The binary summary can be initiated by using *binarySummary* method in spark.ml.classification [145]. A family parameter can be set to opt between LRTS and BLRTS. However by leaving the parameter unset, Spark will take care about the correct alternative between LRTS and BLRTS.
- *Spark 2.3* allows image analysis using deep learning techniques by introducing a new feature known as *ImageSchema* to represent the images in Spark DataFrames, including the utilities to load images from custom formats.
- Moreover, another added new feature in *Spark 2.3* is improved APIs in Python for developers to write the custom algorithms containing *UnaryTransformer* that is useful in writing simple traditional feature transformers and utilities to maintain automatic persistence in loading and saving algorithms [138].

FlinkML

- FlinkML is the new ML library built on top of Apache Flink. FlinkML aims to develop scalable ML algorithms, an intuitive API, and tools that are useful to reduce glue code in ML systems [146]. At present, FlinkML is at the initial stage and the following algorithms were contributed in Flink version 1.3. In supervised learning, *support vector machines*, *Multiple linear regression*, and an *Optimization framework* that are used to solve common optimization problems in ML tasks. In unsupervised learning, *K-nearest neighbors join* algorithm finds exact k-nearest neighbors using quadtree which scales well in a large number of training points. *Polynomial features, standard scaler, MinMax scaler* are useful features in data preprocessing. In recommendation, *alternating least-squares* algorithm factorizes the given matrix A into two factors, a user matrix U and an item matrix V such that $A \approx U^T V$. FlinkML also consists of several distance metrics including Euclidean, Manhattan, squared Euclidean distance, Chebyshev distance, Minkowski distance, Tanimoto distance, and cosine similarity. Moreover, Flink ML is still at an early stage, and there is a wide scope to develop more ML algorithms for linear algebra, clustering, and frequent pattern mining.

ML tools for visualization of Graphs

- *Neo4j Bloom* is newly developed in the 2nd quarter of 2018 by Neo4j graph databases [147]. *Neo4j Bloom* is a graph communication and data visualization tool that enables users to communicate and share their information among them. *Bloom* makes easier for non-developers to explore and make sense of graphs. It allows distinct graph analytics properties on network data. The goal of the *Bloom* is to maintain the existence of a network variations, the changes about how users, devices, data, systems and activities all through the organization are connected. Bloom uncovers hidden relationships and forms abstract graph ideas, so that the users can see and navigate those ideas. Bloom discloses the value of relationships between the data objects, recognizes the connectivity between interesting clusters by identifying hidden paths, connectivity of people to activities, locations, organizations, devices and other objects.
- *Orange* is an open-source tool that allows interactive data analysis to reveal hidden patterns and enable communication between data scientists and domain experts through visualization [148]. Orange consists of many ML algorithms including preprocessing techniques, classification (Naive Bayes, KNN, decision trees, SVM, Regression techniques), unsupervised learning methods (K-means, Hierarchical clustering), evaluation measures (ROC analysis, confusion matrix), dimensionality reduction (PCA), frequent itemset mining and text mining algorithms. All these algorithms work with proper visualizations. Orange also consists of features relevant to visualize complex network data. The core network visualization features are *network analysis*, *network clustering* and *network from distances*. The *network analysis* widget computes node and graph-level summary statistics including degree, average degree, clustering coefficient, density, number of connected components, number of cliques and centrality measures. The purpose of *network clustering* is to find the clusters in a given network data. The clustering method works based on the label propagation algorithm introduced by Raghavan et al. [149] to discover appropriate clusters in the given network. Moreover, the feature *network from distances* builds a network graph for a given distance matrix by connecting the nodes from a given data table. Hence, Orange is a powerful tool to visualize graphs, useful for researchers and scientists to discover the hidden patterns for the given datasets.

3.3.2 Cloud-based machine learning tools

Microsoft Azure ML

- Azure ML Studio is a cloud-based, interactive, drag-and-drop, fully managed service and visual workspace environment to create, test, and deploy predictive analytics tasks for the user's data in the Microsoft Azure cloud.[9] Azure ML Studio comprises of a graphical tool for handling ML tasks, a collection of preprocessing modules, a bunch of ML algorithms and an API model to uncover a model to an application [150]. Azure ML Studio consists of a large number of ML algorithms that facilitates with modules which assist with input data, output, preparation, and visualization. Using these features, a user can develop a predictive analytics evaluation after some iterations and use it to train the users model [151]. The Azure ML has a large number of algorithms which can be categorized into four familiar algorithm families—regression, classification, clustering, anomaly detection, and provides connections to work with problems in *Azure AI Gallery*. Table 5 shows all the ML algorithms developed at Microsoft Azure ML. Azure

[9] https://azure.microsoft.com/en-in/.

ML consists of *Azure ML API* service that provides an Azure ML web service-hosting environment to operate in executing test and production ML experiment evaluations. Azure ML consists of two types of web services. They are *Request-Response Service* (RRS) and *Batch Execution Service* (BES). RRS is a scalable, low-latency service that provides an interface to the stateless models designed and deployed by employing ML Studio. BES is an asynchronous service that scores a batch for data records. The role of scoring is to produce the output which is a part of a predictive web service. However, Azure ML has some limitations. The first limitation is Azure ML does not support for python and R, shortage of staging APIs. Another shortcoming is the Azure ML restriction of dataset size and storage capacity. The data types (Sparse, categorical, strings, and binary data) can be extended to larger datasets during normalization of features and are restricted upto at most 10GB [152].

IBM Watson ML

– *IBM Watson ML* is a cloud-based service that has a hybrid solution connecting user's local environment with the IBM cloud. Watson ML comprises the ML API that is used to publish users models, establish, and score. IBM Watson ML allows users to accomplish two basic ML operations, known as *training* and *scoring*. Training data is used to build an ML model. The method of processing an algorithm by learning a given data is called as *training*. The output of this operation is known as a model. A model encloses the learned coefficients of mathematical expressions. The operation that is used to predict an outcome by using a trained model is called as *scoring*. The central attention of the Watson ML is to deploy a model. However, Watson ML is integrated with IBM SPSS Modeler, Data Science Experience to use ML API to work with ML models and pipelines [153]. Both the software SPSS Modeler, Data Science Experience works with Apache Spark *MLlib* and Python *Scikit*[10] to learn various models taken from ML, data mining and statistical algorithms. IBM Watson Studio, part of IBM Watson, is used to improve the productivity by providing a unique environment where a group of people can work utilizing the best of open source and IBM software to create and deploy an AI solution. IBM Watson Studio constitutes a set of tools for developers, data scientists, research experts to collaborate and work with data to design, develop, and deploy models from that data.

Amazon ML

– *Amazon ML* [154] is a cloud-based, highly scalable ML service embedded with visualization features and wizards to supervise developers via the process of design, develop and train the models. The Amazon ML service uses effective algorithms to develop ML models and discover patterns from the given data. Once the model is ready, Amazon ML uses simple APIs and get predictions for a particular application. *Amazon SageMaker* is a fully managed platform that allows developers/data scientists to create, train, and deploy ML models at a large-scale. *Amazon SageMaker* [155] incorporates both supervised and unsupervised algorithms. Supervised algorithms in Amazon SageMaker include XGBoost (Extreme Gradient Boosting) and linear/logistic regression to solve recommendation and time series prediction problems (See Table 5). In unsupervised algorithms, SageMaker consists of K-means clustering, PCA to obtain the purchasing behavior of customer groups. The other algorithms include a *BlazingText Word2Vec* (Word2Vec algorithm), *DeepAR* (recurrent neural network), *Neural Topic Modeling* (NTM) to learn topics

[10] http://scikit-learn.org/stable/documentation.html.

from text and images, *Latent Dirichlet Allocation* (LDA) to discover the main topic that exists in a text file, a general purpose decoder-encoder algorithm *Seqence2Sequence* and an image classification algorithm is known as *ResNet*. There are certain limits in Amazon ML that the training data file size is restricted to 100 GB. The size of user data files is limited to guarantee that jobs finish promptly on time. At present, the running time for ML jobs is limited to 7 days. If the running time is more than 7 days, it shows a failed status.

Table 5 shows the existing algorithms in different distributed/cloud-based ML tools. We distinguish the ML algorithms by categorizing into supervised (classification, regression), unsupervised (clustering, ARM), dimensionality reduction, recommendation algorithms with additional features. All the distributed ML tools (Mahout, MLlib, FLinkML) developed fundamental algorithms in classification and clustering that are shown in Table 5. Both Mahout and Spark have the parallel FP-Growth algorithm for frequent pattern mining. However, the current version Flink 1.4 does not have frequent pattern mining algorithms. Also, there is a scope to develop algorithms relevant to basic linear algebra, dimensionality reduction, and clustering algorithms in FlinkML. In cloud-based ML tools, Azure ML Studio solves text classification problems using Vowpal Wabbit, a ML library. IBM Watson ML use ML algorithms by working with IBM SPSS and Spark MLlib. Watson ML also consists of deep learning features. In cloud-based ML systems, there is a scope to develop more algorithms in the area of clustering and community detection.

3.3.3 Case studies

In this section, we present some case studies that how the large-scale computing frameworks are useful in applying the ML algorithms to analyze the big data. Mainly, we investigate three case studies known as Netflix, eBay, and Alibaba Group.

Netflix

– *Netflix* [11] is the world's largest subscription service for watching movies and television episodes with more than 100 million members in 190 countries by 2017. Netflix uses Spark Streaming (S2) and MLlib to build user recommendations that are best fit according to customers interests and purchasing histories. Netflix uses live clickstreams and user priorities to modify their recommendation systems every few seconds. In Netflix, everything is a recommendation. Recommendation algorithms are handled by ML algorithms. More than 80 percent of the people are watching movies that come from Netflix recommendations [156]. Apache Spark allows Netflix to use a unified framework/API for ETL, model training and validation. The Spark ML pipeline enables Netflix recommendation pipeline with the necessary steps such as label generation, feature extraction, model training and evaluation to encapsulate as transformers, estimators and evaluators. Hence, the Netflix engineers can build their own *Transformers* (feature engineering), *Estimators* (learning an algorithm) and *Evaluators* (customized measures). These building tools are used to experiment with new pipelines to deploy and produce the results.

eBay

– *eBay*[12] is the one of the largest global market place for online shopping. The enterprise data platform in eBay has more than 200 petabytes and on average 30 terabytes of

[11] https://www.netflix.co.

[12] http://www.ebay.com/.

Table 5 Machine learning algorithms for big data analytics

ML frame works/Systems	Supervised		Unsupervised		Dimensionality Reduction (DR)	Recommendation Algorithms (RA)	Additional features
	Classfication	Regression	Clustering	Association Rule Mining (ARM)			
Mahout [137] version 0.13.0 Framework based	Naive Bayes, Hidden Markov Models (HMM)	Auto correlation, Least squares regression	K-means, Canopy clustering, Distance metrics, Fuzzy k-means Streaming k-means, Spectral clustering	Parallel FP-Growth	Distributed-QR decomposition, Stochastic PCA, Stochastic singular value decomposition (SVD), Lanczos algorithm	Distributed Alternative Least Squares (DALS), Item and user similarity in CF	Topic models (Latent Dirichlet Allocation), Sparse TF-IDF
Spark MLlib [138] Framework based	Support Vector Machines (SVM), Naive Bayes, Random forest and Gradient-Boosted trees, Logistic regression	Linear, Lasso, Ridge and Streaming Regression	K-Means, Bisecting K-Means, Streaming K-means, Gaussian mixures (GMM), Power iteration clustering and Latent Dirichlet Allocation (LDA)	ARM using FP-Growth, PrefixSpan	PCA, SVD	Alternative Least Squares (ALS), Item-/User-based Collaborative Filtering (CF)	Optimization of linear methods- Limited memory BFGS (L-BFGS), Normal equation solver for weighted least squares, Iteratively reweighted least squares (IRLS). Hypothesis—Streaming Significance Testing. Random data generation. Kernel density estimation. Correlation

Table 5 continued

ML frame works/Systems	Supervised		Unsupervised		Dimensionality Reduction (DR)	Recommendation Algorithms (RA)	Additional features
	Classfication	Regression	Clustering	Association Rule Mining (ARM)			
Flink ML [139] version 1.4 Framework based	SVM using Communication efficient distributed dual coordinate ascent (CoCoA)	Multiple linear regression, Optimization framework	K-Nearest Neighbors join	N/A	N/A	ALS	Polynomial features, MinMax scaler Standard scaler, Stochastic outlier Selection (SOS)
Microsoft [151] Azure ML (Cloud-based)	Two-class classification SVM, Averaged Perceptron, Bayes point machine, Decision forest, Boosted decision tree, decision jungle, locally deep SVM and Neural Network (NN).	logistic regression Ordinal, Poisson, Fast Forest Quantile, Linear, Bayesian linear, Neural Network, Decision forest and Boosted decision tree regressions.	K-means	Works with Azure AI gallery	PCA-based Anomaly detection	Bayesian Recommendation systems	Anomaly detection One-class SVM Other Tools Azure AI gallery, a Powerful ML framework Vowpal Wabbit. Text analytics N-gram feature Feature hashing, LDA
	Multi-class classification, Multi-class logistic regression, Multi-class NN, Multi-class decision forest, Multi-class decision jungle, one-v-all multi-class						

Table 5 continued

ML frame works/Systems	Supervised		Unsupervised		Dimensionality Reduction (DR)	Recommendation Algorithms (RA)	Additional features
	Classfication	Regression	Clustering	Association Rule Mining (ARM)			
IBM Watson [153] ML **(Cloud-based)**	Works with IBM SPSS	Linear, logistic loglinear models Generalized Linear Model	Two-step clustering. Works with Spark MLlib	Works with Spark MLlib	Works with Spark MLlib	Works with Spark MLlib	Forecasting Temporal causal modeling. Temporal autoregressive model, Outlier detection. Deep Learning features
Amazon ML Amazon [155] SageMaker (Cloud-based)	XGBoost (Extreme Gradient Boosting), DeepAR, ResNet, an image classification algorithm	Linear/Logistic	K-Means	Works through custom modules in Amazon ML Studio	PCA	Factorization Machines	BlazingTextWord2Vec, Neural Topic modeling to learn topics from texts and images, Latent Dirichlet Allocation, Sequence2Sequence

behavioral and transactional data is extracted on a daily basis [157]. eBay adopts Apache Spark clusters in a virtual environment and uses spark MLlib for providing better customer service by analyzing various applications. The data pipeline in eBay transmits all the activities of the site into the analytics environment by adopting the Apache Spark features. The behavioral data pipeline in eBay consists of all the activities that happen on the Web site including user interactions and clickstream data. eBay uses Kafka for messaging, S2 for stream processing to transform the batch-oriented ETL processes to near real-time infrastructure. The clustering of eBay sellers can be done via K-means algorithm in MLlib. Also, the FP-Growth algorithm in Spark MLlib is useful to find the frequent patterns and association rules from users purchasing history. Moreover, the ALS algorithm in MLlib recommends other items based on their family or friends purchasing history in buying the items.

Alibaba Group

– *Alibaba Group* is the largest e-commerce, retail and AI technology multinational company in China and spread across the globe with more than 50000 employees and 488 million active buyers by March 31, 2017 [158,159]. Alibaba Group has mainly five subcompanies including online global business-to-business trading platform (Alibaba.com), largest commercial center Taobao, online payment services (Alipay), a principal internet portal (China Yahoo!) and a progressive data-centric computing services (Alibaba Cloud) [160]. *Alibaba search* is a personalized search and recommendation platform that is convenient to search and purchase the required products from Alibaba Group, and the search infrastructure team regularly explores the ways to improve the quality of products. To meet the proper search requirements Alibaba works with Apache Flink. Alibaba Group is using a variant of Flink called *Blink* to provide better online services to the end users. Blink runs on a small number of clusters, where each cluster consists of 1000 nodes with high performance for large-scale datasets. Alibaba Group uses Blink to build, train, and update their ranking models in real-time. An issue with Apache Flink is inefficiency in resource utilization. When a Flink cluster starts, it pre-assigns the resources from YARN (refer sec. 4) and later it is troublesome to dynamically resize the Flink cluster which causes to inefficiency of resource usage [161]. To overcome this problem Alibaba Cloud group, made several architectural modifications to coordinate with YARN for accepting dynamical request/release resources. Another issue in Flink is its checkpointing mechanism. Flink has the Chandy-Lamport snapshot algorithm to come at a homogeneous view of the system. However, Flink creates a checkpoint, and it requires to go across the data that exist in all states. But in complex tasks, the state can be bulky and increases over time. It is difficult and inefficient to handle these tasks while producing the results. Blink avoids these type of problems by having an incremental checkpoint that enables to go across new state data only. Flink has two different APIs known as DataSet for batch processing and DataStream API for stream processing. However, Blink unifies both batch and stream processing methods using Table API/SQL by facilitating a unified interface to the users. Hence, Blink works efficiently to create, train, and deploy the ML models in analyzing massive datasets at Alibaba Group using Apache Flink.

– Table 6 shows ML case studies relevant to Spark MLlib and Flink ML based on the parameters domain, goals, ML service, algorithms and other features. Netflix uses ALS algorithm in MLlib for the recommendation of movies by predicting users behavior. eBay uses K-means clustering in MLlib to cluster the products, sellers to provide better customer service. Also, we notice that to find frequent patterns, distributed FP-Growth

Table 6 Some case studies with their corresponding ML algorithms

Company	Domain/ Business needs	Goals	M L Service	Algorithms	Other Service
Netflix	Entertainment	Predicting user behavior	Spark MLlib	Recommendation algorithm-ALS	Storage and Processing via Cloud services
eBay	E-commerce service	Better customer service	SparkMLlib	K-Means clustering	Enterprise Data Platform (Storage, Data streams, Data services)
Alibaba Group	Retail, e-commerce	Better customer service	Flink, Blink	Recommendation algorithms, Naive Bayes, Deep learning	Uses Cloud Computing and Real-time ML platform

from MLlib can be applied to know the users frequent purchasing of products. Alibaba Group has their own cloud computing platform and uses Blink integrating with Flink. Alibaba uses the ALS for the recommendation and Naive Bayes for classification from MLlib.

- Apache Spark empowers the effectiveness of recommendation pipeline in Netflix due to the following reasons. The Rdd-based features such as label generation, feature generation, data retrieval, training, validation using Spark ML pipeline provides a strong coordination across distinct stages of recommendation pipeline and causes outstanding performance gains in Netflix.
- eBay leverages with Apache Spark through Hadoop Yarn. The Rdd-based in-memory computation of Spark results in less overhead, handles high scalability, improves the computational speed and reduces the network I/O cost [162]. eBay trains the k-means clustering model using Spark MLlib to cluster eBay's customers and achieves high performance with respect to scalability and execution time.
- The key developments in Blink runtime environment are *optimized cluster deployment*, *incremental checkpoint design* and *asynchronous I/O features* [163]. In Apache Flink, the cluster deployment consists of a single Job Master and causes to inefficiency when there is a multiple number of jobs. But in Blink's *optimized cluster deployment*, each job has its own corresponding Master. Also, in Blink each job is assigned to its own Task Manager which resolves the collapse of Task Manager in Flink. Moreover, Blink optimizes the prior improvements by introducing a *Resource Manager* which makes coordination with Job Masters and allocates resources in real time for optimal cluster deployment. In Blink's *checkpoint mechanism*, it saves the previous checkpoint data history and stores the updated data at checkpoints. Blink removes the unnecessary data of old state and allows every checkpoint to execute in less time. This causes to decrease the data delays that exist during failover. Another crucial feature of Blink is its *Asynchronous I/O* mechanism, which improves the efficiency of CPU bit computation and enables asynchronous and multi-threaded data retrieval. In this case, the system does not require to wait for sending data to make further requests and manages to send data through a way of callback perspective. Blink buffers all the previous arriving data and wait for the compilation of all successive data and process these data chunks in a batch mode. Hence, the optimized cluster deployment, incremental checkpoint mechanism, and asynchronous I/O feature make the algorithms in Blink efficient in handling real-time computing for large-scale datasets.

3.4 Types of analytics

We classify the analytics by the nature of data, domain and application type (See Fig. 3). If the data type is text, audio, video and web data, then the corresponding analytics are text analytics, audio analytics, video analytics, and web analytics, respectively. Based on the domain type, some of the analytics are Health analytics [164], Financial analytics [165], Social media analytics [166], Urban analytics [167] and Energy Analytics [168]. Graph analytics [169], fraud analytics [170], surveillance analytics [171], risk analytics [172], sentiment analytics [173], streaming analytics [174], sisual analytics [175] are some of the analytics in respect to the nature of the application of various fields of big data. In particular, Graph analytics can be applied to find user's behavior in the analysis of large weblog networks. Some of the applications of big data where analytics can be applied are in the fields of Retail banking,

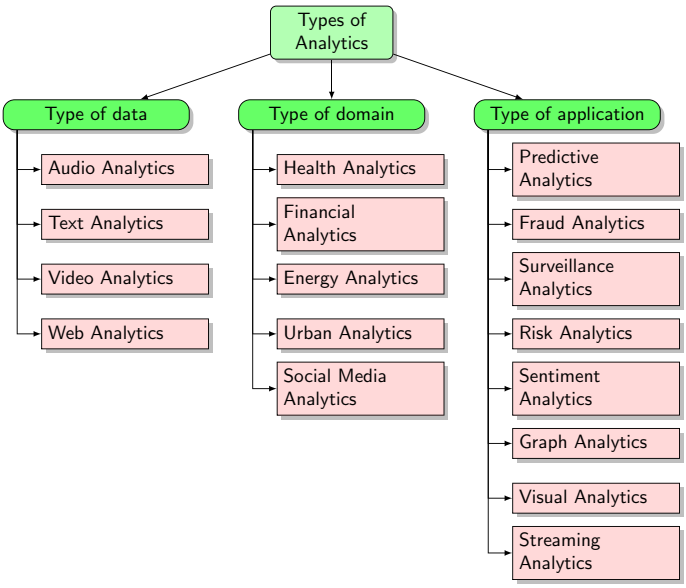

Fig. 3 Classification of analytics

e-Governance, Computational biology, Information and Communication Technology. Table 7 investigates various types of analytics with respect to the techniques and applications.

3.4.1 Visualization

The purpose of visualization is to see the intuitive insights of hidden patterns and to draw some conclusions that are helpful in making faster decisions. Visualize the data patterns to observe uncovered patterns using bar charts, histograms, and pie charts. These are the usual techniques to observe the data in distinct dimensions. The visualizations of large datasets using big data techniques are helpful in decision making to transform data into insights. The main features of visualization tools are statistics (univariate, bivariate and multivariate), data modeling (classification, clustering, predictive analysis), data projection (Principal component analysis, multidimensional scaling) and visual query analysis [175]. The most popular Visualization tools are Tableau, QlikView, JMP, MicroStrategy, TIBCO Spotfire, ADVIZOR and SAS Visual Analytics (VA) in analyzing large datasets.

– *Tableau* consists of VizQL, a query language that converts user interactions into a database query and visualizes it graphically [185]. Tableau supports to visualize data geographically. The strengths of Tableau are a) maintaining speed to analyze millions of rows and produce results in seconds, b) Ease of use based on drag and drop principle, c) Interactive dashboards which allow images, web pages and documents, d) Provides data access to databases, data warehouses and Hadoop and e) The results obtained after analyzation can be shared live on the web and mobile devices. However, Tableau does not have any statistical and predictive capabilities to model the data. It cannot be integrated with social platforms like Twitter, Microsoft SharePoint and other applications. It does not support 3D scatter plots, profile charts, boxplots, heat maps, network graphs, and tree maps.

Table 7 Types of big data analytics

Types of analytics	Techniques	Applications
Text analytics [27,53]	Exploratory analysis, concept extraction, Ontology management, Information Extraction technique, Text summarization (Abstractive & Extractive) techniques, Question-answering techniques, Opinion mining. NLP techniques	Blogs, feeds in social networks, emails, online news surveys and Customer product reviews
Audio analytics [27]	Large vocabulary continuous speech recognition (LVCSR), Phonetic-based approach	Call centers at Business Process Outsourcing (BPO) and Healthcare organizations.
Video analytics [27]	Server-based approach, Edge-based approach.	Recordings of Closed-Circuit Television (CCTV) cameras at shopping centers, video sharing websites (youtube.com)
Social Media analytics [23]	Linear threshold model and Independent cascade model to find the relationships among people. Measures, like degree centrality, Betweenness centrality, closeness, and eigenvector centralities are used in finding the relation between user groups	Social networks (Facebook and LinkedIn), Social blogs (Lifehack), microblogs (e.g., Twitter and Tumblr), crime analysis, marketing, epidemic intelligence, user experience-based visualization. Information diffusion from social networks
Predictive analytics [176]	Regression techniques, Machine learning techniques, Data mining techniques, Auto-coding for clinical text data	Customer relationships, Price optimization, Insurance fraud, Predicting and preventing street crime, domestic abuse and terrorism
Financial analytics [177]	Customer segmentation and profiling, sentimental analysis, clustering techniques, predictive analysis techniques	risk reduction, fraud detection and analyzing fraud patterns, customer satisfaction index analysis, and customer segmentation
Health analytics [73,178]	information extraction, predictive modeling, feature selection, logistic regression, Epistasis modeling for genetic data	case-based retrieval, Heart failure signs and symptoms, patient similarity
Graph analytics [179–182]	Frequent subgraph mining, Closed frequent subgraph mining, Centrality measures, Similarity measures for link prediction, Graph clustering (spectral clustering, random walks, hierarchical methods, modularity-based methods) and indexing. Large-scale graph processing using Pregel, GraphLab, GraphX and Gelly [183]	Anomaly detection in large graphs, analyzing graph structures in streaming data, Extracting communities in web/social networks.

Table 7 continued

Types of analytics	Techniques	Applications
Web data analytics [184]	Tokenization/encoding schemes, PageRank, Hits algorithm, Centrality measures, Finding frequent users using Association rule mining, Markov chains	User navigational behavior analysis, web graph analysis

- *QlikView* The strength of QlikView is, it provides interactive drill-down abilities due to its in-memory architecture [186]. It supports limited statistical and predictive analysis for the given data with some additional upgrades. It also supports to visualize the data using bar charts, pie charts, pivotal and straight tables and multiboxes.
- *JMP* [187] supports heuristic, open-ended, dynamic EDA (Exploratory Data Analysis) features that maintains rich data quality and aggregation steps to view data in different visualization aspects. JMP also supports data to show geographically and facilitates to analyze the linked graph data for several visual analytics.
- *MicroStrategy* [188] constructs dashboards by selecting from a broad range of interactive and advanced visualization features. Users can drag and drop the data to personalize their view or sort, pivot, drill and use aggregations for effective visualizations of input data. MicroStrategy integrates with Google Maps to meet any particular requirements of users for visualizing the data geographically.
- *TIBCO Spotfire* [189] offers analytics for an individual user, cloud-based analytics for the team and big data analytics for various organizations. Spotfire allows to import data and analyze it through data connectors with the Spotfire data engine or to keep the data in the external system. Spotfire allows for interactive and visual analytics for Hadoop and other big data Stores. Spotfire native data connectors comprises of Hadoop data connectors for Apache Hive, SparkSQL, Hive, Impala, MapR drill and Pivotal Hawq. Spotfire consists of ML algorithms like linear/logistic regression, decision trees, random forests, gradient boosting machines and Neural networks.
- *SAS Visual Analytics (SAS VL)* is a web-based Visualization tool meant for exploratory data analysis to uncover patterns in large datasets [190,191]. SAS VL has rich capabilities to analyze and visualize data for several applications in the banking sector, insurance, life sciences, communication, etc. Visual Analytics version 7.1 supports importing data from Pivotal HD and Cloudera Impala. It also allows specifying the schema to import data from ODBC or PostgreSQL data source.
- *ADVIZOR* is a highly powerful and interactive Visualization tool developed at Bell Labs [192]. ADVIZOR is integrated with a columnar in-memory database with high performance. ADVIZOR consists of data transformation tools and provides access to external data sources. It allows predictive modeling without using any code and statistics of data. It has more visual features than Microsoft Excel, more analytics than Tableau, and much simpler than SAS or SPSS (Statistical Package for the Social Science).

In analyzing and visualizing the large network data, some of the graph data visualization software are Cytoscape [193], Pajek [194], NodeXL [195], Gephi [196] and igraph [197]. Cytoscape is used to visualize the biological networks. Pajek and NodeXL analyze the social network data. Gephi is used to analyze and visualize the real-time networks. igraph is useful to analyze and visualize the complex networks. From the above key features of different

Visualization tools, we observe that Tableau is good for visualizing the data geographically and provides access to data warehouses and Hadoop. The drawback of Tableau is, it does not have any statistical predictive capabilities to model the data. However, QlikView supports limited capability of statistical and predictive analysis of data. JMP supports to visualize the data geographically and analyzes the linked data. TIBCO Spotfire supports interactive and visual analytics for big data stores. Spotfire also consists of ML algorithms. Advizor has predictive modeling capability, consists of more visualization features than Excel and more analytics than Tableau.

Summary

In this section, we presented the main components of data analysis stage. The two subprocesses in data analysis are *data analytics* and *data visualization*. In *data analytics*, use the ML algorithms to model the data and analyze the data using data mining techniques. We explored the recently developed framework-based and cloud-based ML tools. We have explored and highlighted the features of Mahout, Spark MLlib, Flink ML with recent developments. We also discussed cloud-based ML tools such as Microsoft Azure ML, IBM Watson ML and Amazon ML. In addition, we compared all the ML systems (both distributed and Cloud-based) in respect of supervised, unsupervised, DR, RA, Additional features and drawn some key conclusions. Moreover, we also investigated case studies, namely Netflix, eBay and Alibaba Group, and explained how ML algorithms are used to analyze large-scale datasets to meet the companies requirements. Furthermore, the classification of analytics concerning to the data type, domain type, and type of application is presented. Finally, we discussed various visualization tools with their strengths and shortcomings.

4 Big data tools and technologies

In this section, we emphasize the widely used big data technologies such as Hadoop 3.0, Hadoop Ecosystem (HE), Spark 2.3 technologies and recently developed infrastructure tools. The predominant big data tools can be categorized as *Batch processing tools*, *Stream processing tools*, *Interactive query processing tools*. However, there exist other subcategory of tools that are relevant to big data infrastructure known as *data ingestion tools*, *Container orchestration tools* and *messaging systems*. We also compare and contrast distinct SQL query tools on Hadoop for complex query analytics in Sect. 4.3. Finally, we elaborate the components of two main large-scale graph processing systems known as GraphX and Gelly. Since it is difficult to compare each category of tools, we compare and contrast the majority of the given tools along with their strengths, limitations and recent developments. This section also emphasizes the concept of *Computing frameworks* (Hadoop, Spark and Flink) as shown in Fig. 2.

- The objective of *Batch processing* is to process arbitrary queries over distinct sets of data. The data can be divided into large batches and achieve latency from minutes to hours.
- In *Stream processing*, the queries can be processed within a progressive time window or on just arrived data record. The data can be divided into independent records or micro-batches of records and achieve latency in seconds or milliseconds.

4.1 Hadoop integrated environment

Apache Hadoop is the popular big data framework which stores petabytes of structured and unstructured data [198]. Hadoop is a MapReduce model-based framework that works in a distributed environment. Particularly, Hadoop is suitable for storing massive volumes of datasets. The fundamental tasks in Hadoop are *storage* and *computation* of data. The responsibility of *storage* can be handled by Hadoop Distributed File System (HDFS) and *computations* are performed by MapReduce framework, respectively.

- *HDFS* is a scalable distributed file system that accepts large-scale datasets to be stored and quickly accessed across large clusters of commodity servers. HDFS governs the storage process of datasets into the Hadoop cluster. It divides large data into blocks and the default size of every block in the Hadoop 3.0 version is 128 MB. For example, if the size of a file is 20 GB, then the file can be split into $20 \times 1024/128 = 80$ blocks. The default replication factor in HDFS is three. Hence, it stores three copies of each block on three distinct nodes in the cluster.
- *MapReduce* is a reliable, fault-tolerant, parallel and distributed programming model to process massive datasets [40,199]. MapReduce performs computational tasks in a batch-wise manner. Instead of moving data to computation, MapReduce works on the concept that *move the computation to the data*. It performs two computational tasks. They are *Map* and *Reduce* tasks. MapReduce accepts the input in (key, value) pairs and after intermediate shuffling provides an output in (key, value) pairs. Figure 4 shows the distinct phases of data flow in the MapReduce paradigm. Also, Fig. 5 illustrates different phases of MapReduce with an example.
  Mapper and Reducer interfaces produce the Map() and Reduce() methods in implementing typical applications. The total number of Maps rely on the input file size and the size of the block. For instance, the standard size of the block is 128 MB, and the input file size is 10 TB, then the total number of maps is equal to $10 \times 1024 \times 1024/128 = 81{,}920$ Maps.

  1. At first, the input file is loaded on to the HDFS. The file is divided into distinct blocks and the size of each block is the same. In Hadoop 3.0, the default block size is 128 MB. The mapper applies Map() method to each record in the data block. The Map() method takes data of each record as (key, value) pairs and process to give intermediate (key, value) pairs.
  2. The method Combine() is optional, which is used for pre-aggregation of the associated key/value pairs to minimize the data transfer cost of the intermediate key/value pairs to Reducers.
  3. The partitioner partitions the mapped output key/value pairs of Map() using a hash function relevant to the keys, *hash(key) mod T*, where T denotes the number of reduce tasks in a MapReduce job [200]. Hence, for a particular job, the cardinality of partitions is equivalent to the cardinality of reducers.
  4. Finally, the intermediate key/value pairs are sorted in shuffle phase and pass to reduce phase.
     For example, (k_n, v_1), (k_1, v_1) (k_2, v_5), (k_2, v_3), (k_1, v_4), $\cdots$, (k_n, v_n) is a chunk of input data (see Fig. 5). The values relevant to the same key are associated with produce intermediate (key, $\langle$ list of associated values $\rangle$) pairs. Then the intermediate pairs are like $(k_1, \langle v_1, v_4 \rangle)$, $(k_2, \langle v_3, v_5 \rangle)$, $\cdots$, $(k_n, \langle v_1, v_n \rangle)$.

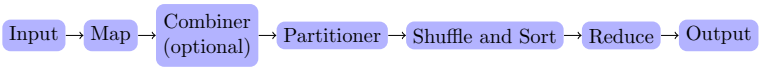

Fig. 4 Data flow in MapReduce

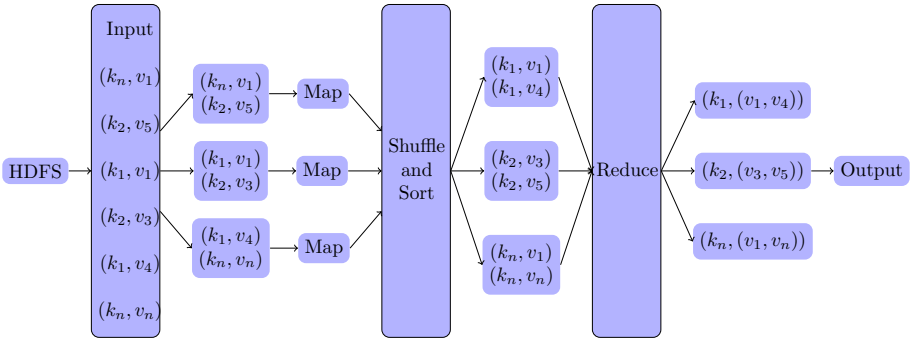

Fig. 5 MapReduce example with different phases

5. The obtained intermediate (key, ⟨ list of associated values ⟩) pairs are submitted to Reduce() function to produce output results.
6. Reduce() methods takes place only after the completion of all Map() operations.

Some potential applications where Hadoop can be deployed are digital marketing automation, the behavioral analysis in retail marketing, mobile device location-based marketing, telecommunications, critical decision making in health care, airline fare management, banking sector, public sector (defense, intelligence agencies), and insurance companies. However, to store, process, and analyze the massive datasets, there is a necessity to develop several data processing tools on the top of the Hadoop. We now present the objective of Hadoop Ecosystem (HE) in the next subsection.

4.1.1 Hadoop 3.0 ecosystem (HE 3.0)

Hadoop Ecosystem is an environment integrated with several required data processing tools which provide services for storage, access mechanism, and data management depending on the requirements of the vendors. It allows synchronization among clients, aggregators, integrators, and data providers [201]. The required data processing tools built on the top of the Hadoop are as shown in Fig. 6.

The Hadoop cluster accommodates a limited number of NameNodes (Master nodes) and more DataNodes (worker nodes). Hadoop distributed file system works on master/slave perspective. The Master node comprises a *NameNode* and a *JobTracker*. The components of a slave node are *DataNode* and *TaskTracker* [17]. The master node consists of metadata of a file to be stored in HDFS. Metadata contains the details about blocks concerning the file and their actual locations on the *DataNodes*. The *JobTracker* takes the responsibility of scheduling a job by performing individual Mapper and Reducer operations to individual nodes. It also handles failed tasks by rescheduling the job. In HDFS, each *DataNode* stores the actual blocks of a file on its own local disk. The *TaskTracker* takes the responsibility of MapReduce operations of their corresponding *DataNodes*. However, the limitation of Hadoop 1.x is the single point failure of Namenode (SPOF) in HDFS cluster [202,203]. The NameNode fails due to the following reasons. The first one is, an unexpected machine crash causes to the

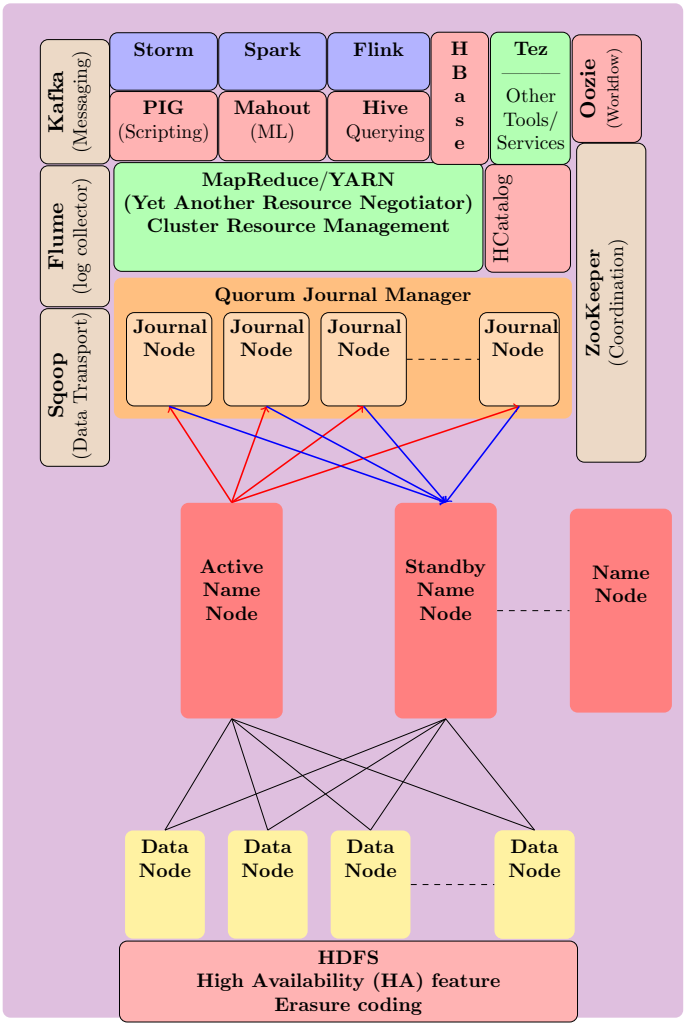

Fig. 6 Hadoop 3.0 ecosystem

failure of NameNode, and the second reason is, due to planned events like hardware or software upgrades on the NameNode results in cluster downtime. If the NameNode goes down, the entire cluster is unavailable, until an operator restarts the NameNode or change it on to a separate machine.

– Hadoop 2.x overcomes the limitations of Hadoop 1.x by providing the High Availability (HA) architecture in HDFS. The HA feature consists of two NameNodes. They are Active NameNode (ANN) and Standby NameNode (SNN). The ANN takes the responsibilities of accepting all jobs from clients and the SNN synchronizes with the ANN to provide a fast failover operation if required. The Quorum Journal Manager (QJM) is used to configure the HDFS HA feature to share edit logs between the two NameNodes [204]. The ANN and SNN communicate each other using a group of daemons known as JournalNodes (JNs) (See Fig. 6). If any namespace modification is accomplished by the ANN, it firmly

logs a record of the modification to the JournalNodes. Then the SNN reads the edits from the JNs and regularly observes them for changes to edit the log. Also, SNN copies the modified edits to its own namespace. In the case of failover, the SNN read all of the edits from JNs before converting itself into ANN.

– The next generation MapReduce, which is MRV 2.0 (MapReduce version 2.0) is YARN. The goal of YARN is to divide the responsibilities of TaskTracker, resource management, job scheduling, and supervising into different entities [205]. The objective is to possess a global ResourceManager (RM) and ApplicationManager (AM) per-application, a slave per-node NodeManager (NM), and a Container per-application running on a NodeManager. The purpose of RM is to coordinate all the resources of the applications in the system. The RM, slave per-node and NM form a framework for data computation. The ApplicationMaster (AM) handles the resources to monitor and execute the jobs by working with NM. NM takes the responsibility of containers, monitors the resource usage and communicate the same to the RM.

HE 3.0 features

– *Erasure Coding (EC)*
Rather than using replication, HDFS in HE 3.0 supports *Erasure Coding (EC)* to reduce the storage space significantly [206]. The replication factor in HDFS is three, which is more expensive. The 3× replication method in HDFS causes to 200 % overhead in storage space and network resources [207]. Hence, instead of using replication, a better approach is to use *EC* method, which ensures the same degree of fault-tolerance with limited storage space. For example, a 3× replicated file with 10 blocks requires 10 × 3= 30 blocks of space. But, using *EC* method, (10 data, 3 parity) will consume only 13 blocks.
– *Multiple NameNodes*
Another improved feature in Hadoop 3.0 is it supports to use *multiple NameNodes*. Hadoop 2.0 has an ANN, a single SNN and achieves fault-tolerance by replicating edits to a quorum of three JNS. Hadoop 2.0 is capable of handling failure of any node in the entire system. However, to achieve a higher degree of fault-tolerance for some particular deployments, Hadoop 3.0 maintains multiple standby NameNodes [207].
– *Data Lake and Aliyun*
Hadoop 3.0 supports the alternative Hadoop compatible file systems: Azure Data Lake[13] and Aliyun Object Storage System (Aliyun OSS),[14] a file system of Alibaba Cloud.
– *Intra-DataNode balancer*
Hadoop 3.0 consists of another important feature known as *Intra-DataNode balancer*, which handles the skewness while adding or replacing the disks in DataNode.

Various data processing tools in *HE* that are used for the analysis of massive datasets are HBase, Hcatalog, Hive, PIG, Mahout, Sqoop, Flume, Oozie and ZooKeeper. We explore the majority of the tools of *HE 3.0* in each subsection of Sect. 4. These tools can query over the voluminous datasets with good performance in less time. *HE* provides an access mechanism to store, process and analyze big data. *HE* consists of both SQL and NoSQL tools to handle large-scale structured and unstructured datasets in managing the data. In the next subsections, we present Spark 2.3 and compare distinct SQL tools on Hadoop for querying on large-scale datasets.

[13] https://azure.microsoft.com/en-in/solutions/data-lake/.

[14] https://www.alibabacloud.com/product/oss.

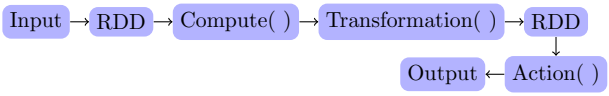

Fig. 7 Data flow in Apache Spark

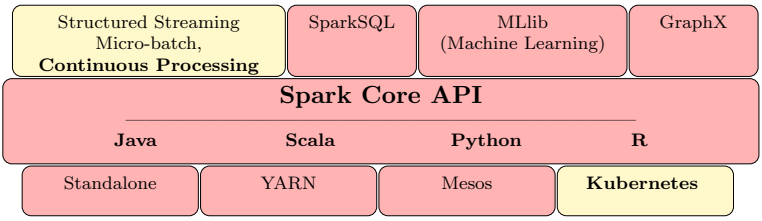

Fig. 8 Spark 2.3 ecosystem

4.2 Apache spark 2.3

MapReduce-based platforms are suitable for processing large-scale applications on commodity clusters. MapReduce-based systems follow the acyclic data flow model, and hence, these systems are inefficient for handling iterative tasks across parallel operations. Apache Spark is an in-memory, fast cluster computing framework written in Scala, that processes both the iterative and interactive jobs by achieving high scalability and fault-tolerance [208]. Spark is compatible with Apache Hadoop and works with any Hadoop-supported storage system. Spark is in-memory, which is an added advantage that the fine-grained control over the intermediate data storage can be done with changing I/O specifications [209]. The two major components in Spark are, one is creating *Resilient Distributed Datasets* (RDD) and the other is *parallel operations* to be performed on the datasets. An RDD is a set of elements split-up over a set of nodes which can be performed on in parallel. Figure 7 illustrates the data flow in Apache Spark.

The compute() method in Fig. 7 reads the input data from previous RDD and then apply a transformation() to produce new output results. A transformation() can be obtained by applying a series of transformation operations (map, filter, groupBy, join, etc.) on RDD. A transformation() can produce a single RDD or multiple RDDs depending upon processing steps. An Action() returns a value after applying the transformation on the dataset. Spark combines the libraries like SQL, Machine learning library (MLlib), GraphX [181] and Spark Streaming. Spark can run on Hadoop, Mesos and access data from different sources like HDFS, S3 and HBase [142]. Apache Spark version 2.3 and Kubernetes [210] integrate their features to maintain a distributed processing environment and orchestration at scale. Figure 8 illustrates various tools including new components *Structured streaming* and *Kubernetes*. In the current version Spark 2.3, users can start Spark workloads intrinsically on a *Kubernetes cluster* and utilize the *Kubernetes scheduler* as the backend to attain superior resource usage and multi-tenancy by allowing workloads [211]. The *structured streaming* process consists of a new feature known as continuous processing execution mode. We explain this feature in Sect. 4.4.

4.3 SQL querying tools on Hadoop

Many users and enterprisers are comfortable querying with SQL. The query processing using SQL on the data located in a distributed file system (HDFS) has a crucial job in big data

analytics. SQL tools on Hadoop are efficient for handling interactive, ad hoc and complex queries on large-scale datasets. Hence, several SQL tools querying on Hadoop tools were developed in the recent years. Table 8 emphasizes the differentiation among various SQL query tools on Hadoop in respect of ten parameters. They are ACID (Atomicity, Consistency, Isolation, Durability), OLAP operations, Hadoop environment, Machine learning library, Business Intelligence tools, Data Analysis, Visualization, file format, Relational and NoSQL databases. These parameters are useful in finding the efficiency of SQL querying tools on Hadoop.

- The tools Hawq, Hive, Presto, Vertica and Hadapt satisfies ACID properties, whereas Goolge Tenzing does not support isolation. The tools which perform OLAP operations are Hawq, Hive, Kylin and Tenzing.
- All the query tools in Table 8 support Hadoop MapReduce.
- The tools Hawq, SparkSQL, and Apache drill support to do ML on Hadoop.
- Clients can access Hadoop data using Kylin that performs well than Hive queries. The query tools that are compatible with Visualization tools are Hawq, Hive, SparkSQL, Presto, Drill, Kylin, Vertica and Hadapt. Hawq, Hive, SparkSQL, Apache Drill, Kylin, HP Vertica are compatible with Business Intelligence, data analysis and Visualization tools. However, Hadapt and Impala support business intelligence tools, but they do not support data exploration features.
- Impala, Presto and Apache drill use PostgreSQL to execute query fragments. These tools can also be accessible to non-relational databases. Hive, SparkSQL, and Apache Drill are accessible to query the HBase data. Presto queries the data from the NoSQL database, whereas Cassandra and Apache drill query data from MongoDB database. Various supported file formats of the mentioned SQL Query tools on Hadoop are shown in Table 8.

From Table 8, we observe that Pivotal HD Hawq, Apache Hive, SparkSQL, Apache Drill and Kylin satisfy majority of the properties related to the above-mentioned parameters.

4.4 Stream processing systems

Stream processing systems play a crucial role providing valuable information to customers and empowering industries to take appropriate and quick decisions. Hence, **S**tream **P**rocessing **a**s **a** **s**ervice (SPaas) offering. Stream computing tools process the live streams of data and produce low latency results. Applications required for stream processing are generally denoted in the form of directed acyclic graphs, where the vertices denote as operators and edges represent as data flow channels between operators [222]. In this section, we discuss some widely used distributed stream processing tools including cloud-based streaming systems with recent developments. A comparison of distributed stream processing tools is given Table 9.

4.4.1 Distributed stream processing systems

Apache Storm

- *Apache Storm* [223] is a distributed, fault-tolerant, real-time computing system for massive volumes of streaming data. The core feature in Storm is a *stream*. A *stream* can be a continuous ordered list of tuples which can be computed in a distributed approach [224]. A stream is represented as a schema that consists of information about distinct fields (

Table 8 Comparison of SQL query tools

	ACID	OLAP	Hadoop Environment	Machine learning library	ODBC/JDBC BI tools	Data Analysis	Visualization	File format	Relational databases	NoSQL database
Pivotal HD HAWQ [37]	✓	✓	Support	✓	Both	✓	✓	Parquet	SQL compliant	X
Apache Hive [212]	✓	✓	Support MapReduce and Tej	X	Both	✓	✓	Parquet, text, rc file	✓	Support HBase
Cloudera Impala [213]	X	X	Support	X	Both	X	X	Avro, RCFile, text	MySQL/ PostgreSQL	X
SparkSQL [214]	X	X	Support HDFS	✓	Both	✓	✓	JSON, Parquet, Text, Hive file formats	Compatible with hiveQL	Support Hbase
Facebook Presto [215]	✓	X	Access to Hadoop	X	JDBC	X	✓	JSON	MySQL and PostgreSQL	Access to cassandra
Apache drill [216]	X	X	support	✓	Both	✓	Tableau	JSON, Parquet, text	(Not)Schema-free	Support MongoDB, Hbase

Table 8 continued

	ACID	OLAP	Hadoop Environment	Machine learning library	ODBC/JDBC BI tools	Data Analysis	Visualization	File format	Relational databases	NoSQL database
Apache Kylin [217] [218]	ANSI SQL	✓	Support	X	Both	✓	Tableau,	Ajax	MySQL	X
HP Vertica [219]	Support ACID ANSI SQL	X	Access to HDFS and hive	X	Both	✓	Tableau,	Parquet, ORC supports joins, complex data types	✓	X
Google Tenzing [220]	Does not support isolation	✓	support	X	X	X	X	Structured data	support SQL	X
Hadapt [221]	SQL	X	MapReduce	X	Both	X	Support	Query JSON, XML, HL7, log files	PostgreSQL	X

e.g., integer/float/double, long/short, strings, boolean) in tuples. Apache Storm comprises mainly three abstractions. They are *spouts, bolts and topologies*. A *spout* is an origin of streams in a computing process. A *spout* creates its own streams or reads the data from messaging tools (e.g., Kafka, RabbitMQ) or produce streams by reading the data from external streaming APIs. A *bolt* computes multiple input streams and generates multiple output streams. A *spout* sends the data to one or more *bolts*. A *bolt* receives data from *spout* and process the selected number of data streams to produce new data streams [225]. A network of spouts and bolts is known as a *topology*, where every node is a spout or a bolt. An edge in the topology network denotes a bolt subscribing to output data streams of another spout or bolt. In Storm, first topologies can be defined and then according to topologies a stream of incoming data can be run continuously. A topology is a graph of computation. Spouts read the information in tuples from a source and send the data to a topology using the messaging system Kafka. A cluster in the Storm is alike to Hadoop cluster. In Hadoop, *MapReduce jobs* can be run, whereas on Storm, *topologies* can run. The distinction between *MapReduce job* and *topology* is that *MapReduce job* completes after processing, whereas *topology* processes messages forever. A Storm cluster comprises of master and worker nodes [226]. Similar to Hadoop JobTracker, the master node in Storm runs a daemon known as *Nimbus*. In Storm, only *Nimbus* is the master node and rest of the nodes are worker nodes. Each slave node runs a daemon called the *Supervisor*. Zookeeper maintains the coordination between *Nimbus* and *Supervisors*.

  - Apache Storm achieves fault-tolerance when a worker or a node or a *Nimbus*/*supervisor* fails. When a worker node fails, the *Supervisor* again starts the worker node. If the worker node fails to heartbeat to *Nimbus* successively, the Nimbus reallocate the worker node to a different machine [227]. When any uncommon problems are experienced, the *Nimbus* and *Supervisor* daemons fail or die, they start again by assuming nothing has occurred as they acted as stateless and modeled as fail-fast. In Hadoop, if the JobTracker breaks down, all the running jobs will be lost, whereas in Storm, the failure of *Nimbus*/*Supervisor* does not effect the worker nodes. If the worker nodes die, the supervisors will restart them. Moreover, without *Nimbus*, a worker node could not be reassigned to some other machine. Hence, Nimbus act as a kind of Single Point Of Failure (SPOF).
  - The latest version Storm 1.2.0 consists of improvements to the integration of Kafka to Storm which enhance stability, ease configuration. The improved metrics that are used for internal reporting statistics are acknowledged, failed, transferred and disruptor queue metrics [228]. Moreover, the new features of Storm 1.2 support HBase as state backend, bolt-spout memory configuration, no acks/fails for *KafkaSpout*, and governing the tuple state modifications to *kafkaSpout* by creating a listener.

Apache Samza

- **Apache Samza** is a fault-tolerant stream processing computing framework in a distributed approach, built on top of Apache Kafka for messaging and Hadoop YARN for resource management. A Samza stream is a partitioned, ordered, multiple-member (subscriber) chain of messages with no loss of data [229]. Also, streams in Samza buffers that segregate computing stages from each other. A stream in Kafka is a topic, and in a database, a stream can be read by consuming modifications to a tabular data. Samza integrates with pluggable systems that consist of stream abstraction features. In order to maintain high throughput, each stream is divided into a single or multiple partitions, where every partition is an ordered sequence of messages. Samza works in an integrated environment with YARN and Kafka. The three layers for execution, streaming,

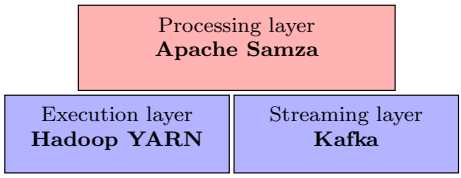

Fig. 9 Layers in Samza

and processing are shown in Fig. 9. Figure 10 illustrates the architecture of Samza with various components. Different colors in Fig. 10 indicate distinct host machines. The main components of YARN are ResourceManager (RM), a NodeManager (NM), and an ApplicationMaster (AM). In the cluster, every node has a NodeManager. The role of NM is to initiate the processes on that node (machine). The RM communicates with all NodeManagers and says about which application has to run. The Samza client starts a new Samza job by talking to the YARN RM. A Samza job consists of a set of input, output descriptions and a *StreamTask* which is to be performed for every input message. A *StreamTask* instance is a fundamental class on which Samza jobs are to be executed. The YARN RM communicates with NM to assign memory for Samza's ApplicationMaster (Samza AM). Samza AM requests YARN RM for YARN containers to execute *SamzaContainers*. The responsibility of *SamzaContainer* is to manage the start, process and turn off of *StreamTask* instances.

A Samza job comprises various *SamzaContainers* running on distinct nodes. Samza *StreamTasks* take the input and produce an output through kafka brokers that are placed in the same nodes or machines. When a Samza job starts, a set of multiple *StreamTasks* are generated and supplied to *TaskRunners*. The *TaskRunners* characterize the execution context of the task instances which are handled by the runtime environment of Samza [230]. Samza works with Kafka for messaging which ensures the delivery of messages. When the messages are on a waiting list and the queue is not empty, then the following issues may occur. They are a) Dropping messages causes to inaccurate results, b) log the messages to disk causes to loss of data, and c) blocking the messages leads to the whole system can come down. Samza overcomes these issues by using Kafka's message broker as a remote buffer. In the case of message blocking, the consumers and producers are separated by utilizing Kafka's broker's disks as buffers. Kafka's message brokers are highly available in the case of message dropping (version 0.8 onwards). In the case of event failure, Samza job reestablishes on another node or machine, where there is no loss of input/output data, since the Kafka brokers stores the data by maintaining replication. Samza does not maintain a remote database for data storage, instead every Samza task maintains a key-value store that occurs in the same node. The cluster maintains the replication by copying the modifications made to the key-value stores in other nodes. However, if one node fails, Samza works with the YARN, and the current state of the tasks can be restored in another node. The limitation of Samza state management is, currently (version 0.14.0) it supports at least once delivery, but it does not support exactly once semantics. In at least once method ensures the processing of every message, but in this method, duplicates may exist. Another limitation is, sharing the data between tasks (across partitions) require some extra effort and consumes more space for repartition and distribution of data.

Simple Scalable Streaming System (S4)

– **S4** is a general-purpose, distributed stream processing engine motivated by MapReduce model and developed at Yahoo [231]. It is a scalable, partially fault-tolerant, pluggable

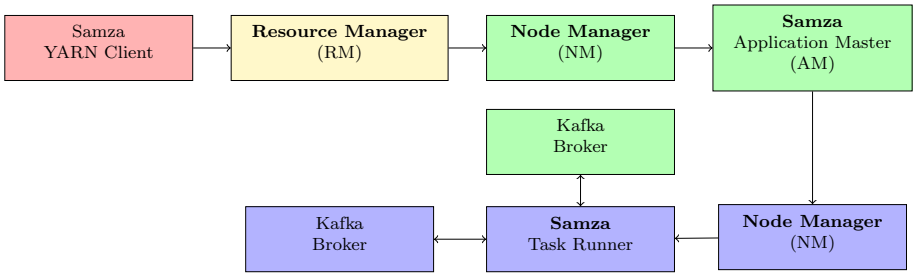

Fig. 10 Architecture of Samza

platform that is used for handling ceaseless and boundless streams of data. The basic computational units in S4 are *processing elements* (PEs) distributed over the cluster. The message transfer among the PEs will occur in the form of data events. An instance of a PE is recognized by the components known as *functionality, type, keyed attribute* and *value of a keyed attribute*. Each PE ingests the events corresponding to the value on which it is keyed. *Processing Nodes* (PNs) are logical hosts to PEs, which are accountable for listening, processing, sending and producing output events. S4 achieves partial fault-tolerance, if there exists no user-specified state/message fall back function in PEs, then the states related to failed PEs and information has vanished during the handoff.

Spark Streaming (S2)

– *S2* [232,233] is a reliable and high-level abstraction for processing various streaming applications. Most existing conventional stream processing systems work based on continuous operator model (COM). The COM model works on the principle that process one streaming data record at a time. The COM is simple and works efficiently for small-scale data and does not perform well with large-scale datasets [63]. To overcome the challenges faced by COM model, Zharia et al. [232] proposed a new high-level programming abstraction known as *Dstreams* (discretized streams). A *Dstream* is a continuous sequence of data that arrive at a particular time interval. Technically, a *Dstream* consists of a sequence of RDDs which contains data at a particular time interval. The core features of S2 are *Sources*, *Input DStreams*, *Receiver*, *StreamingContext*, *Transformations* on DStreams and *Output operations* on DStreams. S2 allows real-time data as input streams and split-up the data into batches. Thus, the Spark engine processes the batches of data to produce output streams in batches [234].

  – The main feature *StreamingContext* creates an object which initializes the *Spark-Streaming task*. After creating a context, S2 doesn't allow for new streaming computation at the same time. Only if the previous StreamingContext task is finished then the next *StreamingContext* object can be initiated.
  – *Checkpointing*
  Checkpointing is a method to write the received input DStreams at checkpoint intervals to suitable hdfs-compatible storage. Checkpointing permits to create stream processing pipelines which are fault-tolerant in such a way that it restores the input DStreams in the case of failure occurs. S2 involves two types of data which need to be checkpointed. They are *Metadata* and *Data checkpointing*. *Metadata* comprises the configuration details which are used to generate a streaming application, DStream operations and batch jobs which are in a queue but not yet finished. The *Metadata checkpointing* saves the details of performing a streaming process to a fault-tolerant

storage system (HDFS). The *Data Checkpointing* saves created RDDs to a storage system, which is reliable and essential in some stateful transformations that merge the data from multiple batches. While using the transformations, the obtained RDDs rely on RDDs of preceding batches that forms a dependency sequence which is growing in respect of time. To break such a dependency sequence, the transformations are to be checkpointed to a well-established storage system (HDFS) at regularly occurring time intervals.

– *Write-ahead-logs (Journaling)*
In S2, the data received from advanced sources are buffered in the memory of the executor as far as the processing has been completed. If there is any failure in the driver process, the buffered data can be lost, and it cannot be retrieved even after the driver has been restarted. To overcome this limitation, Apache Spark version 1.2 onwards (current versions 1.2, 2.2, 2.3) introduced a new feature known as *write-ahead-logs* or *journaling* in S2 [235]. *Write-ahead-logs* ensures the durability of data operations by first writing the operations to a durable log and then applied to the data. If the system fails while applying the operation, it can be recollected from the log, and again the operation can be performed on the data. S2 maintains fault-tolerance by facilitating *write-ahead-logs* in such a way that the received data is also stored in the log files in a fault-tolerant file system.
– Along with micro-batch streaming, the current version *Apache Spark 2.3* consists of *structured streaming* that has an additional new streaming query execution feature known as *continuous processing execution mode* (CPE) (See Fig. 11). The CPE mode can execute streaming queries with sub-millisecond (1 ms) end-to-end latency and guarantees *at least once* delivery mechanism [236]. Unlike in MEP mode, rather than launching periodic jobs, the *Spark 2.3* driver initiates a set of multiple long-running tasks at the start of the query that read, process and writes data events continuously. The S2 pipeline receives the data events that are available in the source and process to write the events to the sink with end-to-end latency (1 ms) in a few milliseconds. However, the earlier version of Spark 2.0 has only structured streaming execution mode in which micro-batch processing can be used for streaming queries with lower (100 ms) end-to-end latency and guarantees *exactly once* delivery mechanism. The improvement of a streaming query will be checkpointed by using prominent *Chandy-Lamport* algorithm [237], which is a snapshotting mechanism that is effectively used in distributed systems for maintaining a consistent record of the global state of an asynchronous system. In CEP mode, there exist some particular marker records known as *epoch markers* that are to be inserted into the input data stream of each task. The space between the epoch markers is called as *epochs* [238]. When a task in CEP encounters a marker, the task asynchronously reports the final offset and handovers it to the driver. Similarly, the driver receives the offsets of multiple tasks and while writing them to sink, it writes them to aforestated write-ahead-log. As the mechanism of checkpointing is fully asynchronous, the tasks can prolong incessantly and yield end-to-end millisecond latencies. *Structured streaming* in Spark 2.0, supports joins between a streaming Dataset and a static one, whereas Spark 2.3 introduces another interesting feature known as *stream-to-stream joins* (inner/outer joins), the ability to join two different data streams, buffering rows up to matching tuples appear in another stream.

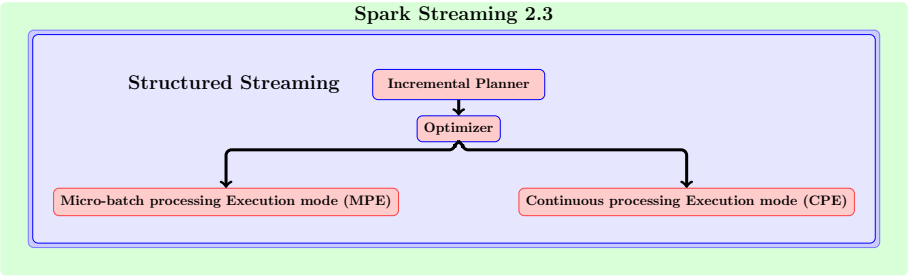

Fig. 11 Spark Streaming 2.3 architecture

Apache Flink

– *Apache Flink* [24,146] is a data flow framework for distributed batch and Real-time stream processing for massive datasets. Flink was originally developed as a Stratosphere for the analysis of massive datasets and then incubated as an Apache Project in April 2014 [239]. The layered architecture of Flink consists of APIs for batch (*DataSet API*) and stream (*DataStream API*) processing, and the core component *Runtime*, a distributed data processing engine which collects the program. Flink can be deployed on a local machine using JVM (Java Virtual Machine), in a cluster managed by YARN or Mesos and run on a cloud (e.g., EC2). The *Runtime* engine of Flink processes the data on the basis of an event at a time instead of sending data as a sequence of micro-batches. Flink supports connectors for input and output of data streams. A connector is an interface which connects with third-party projects (e.g., Apache SAMOA,[15] Zeppelin [16]). Flink is compatible with the connectors such as Kafka (sink/source), HDFS (sink), Redis (sink) and Flume (sink). Flink collects the data from a source and outputs through a sink.

  – The fundamental components in Flink are streams and the transformations which are to be performed on streams. A stream is an intermediate result, and a transformation is an operation which takes input as single or multiple streams and gives the output regarding one or more streams.
  – The *DataSet* is a Flink regular program that runs transformations (e.g., filter, map, join, group) on data sets. A *DataSet* in Flink is internally represented as a stream of data. *DataStream* is a Flink regular program that runs transformations on data streams (e.g., flatMap, updating state, WindowReduce, WindowFold, aggregation).
  – The Flink *distributed Runtime environment* consists of two processes. They are *Master processes* or *JobManagers* and *Worker processes* or *TaskManagers*. The *JobManagers* handle the responsibility of scheduling tasks, coordinating checkpoints and recovery in case of failures. The data flow in Flink is as shown in Fig. 12. A highly available setup may have several *JobManagers*, out of which only one will act as a master and the rest are standby [240]. Each TaskManager is a JVM process and performs single or multiple subtasks in separate threads.
  – Flink maintains a *fault-tolerance* mechanism using the concept of *checkpointing* (checkpointing of file state). A *checkpoint* is concerned to a point in every input data stream and enables to a particular point to recover state and positions in the streams to provide the same semantics to the application, so that it executes without

[15] https://samoa.incubator.apache.org/.

[16] https://zeppelin.apache.org/.

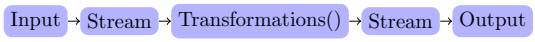

Fig. 12 Data flow diagram for Apache Flink

failures [241]. Flink requires knowing the perception of the state to achieve state as fault-tolerant using *checkpoints* and *savepoints* of streaming applications. The state of a stream processing is represented as [*Source* (source event sequence per event), *Pos* (last processed position)], *S_SYS* (internal status of stream processing engine) and *S_OUT* (emitted). A snapshot of a stream processing system can be denoted as *([Source, Pos], S_SYS, S_OUT)* at a particular point of time [242]. Programs in DataStream API can resume processing at a save point. *Savepoints* are manually triggered checkpoints that capture a snapshot of a program and write it to a State Backend. A particular State Backend implements the internal representation of a state and the state persistence upon checkpoints. *Savepoints* are self-contained checkpoints that are saved externally and can use stop-and-resume or update Flink programs. The mechanism of checkpointing in Flink works on the concept of consistent snapshots to ensure exactly once semantics. Each function and operator in Flink are stateful. A stateful function stores the data over processing of independent events to make state as an essential building block to handle more complicated operations. At the time of execution, programs are snapshotted on worker nodes at regular intervals and provide checkpoints. To recover the data, only the final checkpoint is required and the previous checkpoints are to be removed as soon as a new checkpoint is finished. Flink offers both low-latency stream processing with support for traditional batch tasks. Flink processes batch programs as a particular case of streaming programs, where the streams comprises of finite number of elements. Flink is probably best suited for organizations that have heavy stream processing requirements and some batch-oriented tasks. Its compatibility with native Storm, Hadoop programs and its ability to run on a YARN-managed cluster can make it easy to evaluate.

Scalable Advanced Massive Online Analysis (Apache SAMOA)

– *SAMOA* [243] is a framework that comprises a programming abstraction for distributed streaming ML algorithms. SAMOA is simple to use and constitutes a set of distributed streaming algorithms for the fundamental ML algorithms like classification, clustering, and regression. The architecture of SAMOA has an advantage that it has a plug-in facility that enables to execute on Storm, Samza, Flink and S4.

  – *SAMOA* describes its own topology and the basic elements in a topology are a *Processor*, *Content Event*, *Stream*, *Task*, *Topology Builder*, *Learner* and *Processing item*. A *Processor* is the fundamental logical computing unit which consists of all logical codes. A *Content Event* is an event that comprises the content which is to be handled by the processors. Multiple processors are connected by streams, where a stream is a physical unit of a topology. A stream has a particular source, but it has multiple destinations. A *Task* is alike a Hadoop job. A topology can be described as an internal part of a task. *TopologyBuilder* is a class that constructs all the physical components of the topology and combine them. A *Learner* in SAMOA is designed as a sub-topology. In fact, *Learner* is the fundamental interface that plays a major role in learning algorithms. An algorithm in SAMOA can be designed using the logical units of *Processor* and *Content Event*, whereas *Stream/Task* are the physical units to write several segments of the algorithm [244]. A *TopologyBuilder* acts as a supervisor

in managing all the bookkeeping (maintaining the records of various applications) services.
- An advanced feature in SAMOA is a *Processing Item* (PI), which is an invisible topology unit, and exists internally in the topology. A PI does not contain any logic; instead, it makes connectivity between a Processor with all other processors in the topology. Another processing item in SAMOA is *Entrance Processing Item* (EPI). An EPI acts as a source of a topology and creates its own streams. EPI has connectivity with external sources, drags the data and sends the data streams to the topology.

The current version SAMOA 0.4, includes VHT classifier (Vertical Hoeffding Tree), CluStream (clustering), Adaptive Model Rules (AMR-Regression) and PARMA (on going work for frequent pattern mining) [243]. VHT is a distributed classifier that employs vertical parallelism on top of Very Fast Decision Tree (VFDT), a standard streaming algorithm. CluStream is a distributed clustering algorithm for stream datasets. AMR is an innovative regression algorithm for learning the streaming data using the vertical parallel implementation in SAMOA. PARMA is a basic algorithm for mining frequent itemsets using micro-batching and distributed approach [245]. The developers can design their own new algorithms and can run on these distributed streaming engines. Currently, SAMOA project is under incubation to build and add new algorithms in SAMOA.

Google MillWheel

- *Google MillWheel* is a scalable, low-latency and fault-tolerant complex stream processing framework developed at Google [246]. MillWheel is a graph computation of user-specified transformations on input data and emits results after processing. The input/output data constitute a set of triads (*key*, *value*, *timestamp*), where the *key* is the metadata of semantic relations in the system. The *value* field represents a byte string of a particular record. The *timestamp* (event time or ingress time) in the triad is a value near to clock time set by users of MillWheel. The timestamps are useful in finding the low watermarks. A *low watermark* is a tight bound on the *timestamp* of total events that exist in the streaming system. The core features in MillWheel are *computations*, *keys*, *streams*, *persistent state*, *low watermarks* and *timers*. A *computation* in MillWheel is the logical code of an application defined by a user. When the input data enters the MillWheel system, the computation code invokes data and the user-specified operations are activated. *Keys* are the fundamental abstractions for comparing and grouping distinct records. For each record, a consumer defines an *extraction function* that allocates a key to the record. Streams are useful in carrying the data between distinct computations. A computation in MillWheel ingest input data streams and publish the output data streams. Every consumer specifies key-extraction functions per stream so that multiple consumers can subscribe to a specific stream. A *persistent state* is a byte string that is assigned for every key. The user supplies serialization/deserialization procedures to interpret mechanisms like protocol buffers that occurs. The *persistent state* can be maintained by getting back a highly available data store (BigTable) that guarantees the integrity of the data. Google MillWheel ensures the fault-tolerance by having *exactly once delivery* feature according to the following properties. i) duplicates are eliminated, senders acknowledged, ii) carrying messages are at rest until they are acknowledged to maintain at least once delivery mechanism, and iii) total records have corresponding unique_Ids. MillWheel computes streaming joins for several Ads customers, who require low-latency improvements to client-visible dashboards. However, MillWheel does not suitable for the applications which are not parallelized well among distinct keys.

Twitter Heron (TH)

– *TH* [247] is a real-time stream processing system in a distributed approach developed at Twitter. *Heron* supports for at most once, at least once, and exactly once processing delivery mechanisms. At first, the Twitter uses Storm for real-time stream processing. But later they developed *Heron* to process the large-scale real-time data with better performance, by overcoming the limitations of Storm. The objective of *Heron* is to preserve the compatibility with the Storm API. *Heron* works based on the concept of topologies which are similar to Storm topologies. A *topology* in *Heron* is a directed non-cyclic graph computation of *spouts* and *bolts*. Also, a *topology* is similar to a logical query in database management systems. The core features in *Heron* architecture are *Aurora scheduler* [248], *containers*, *Topology Master* (TM), *Stream Manager* (SM) and *Heron Instance*. The users initiate the *Heron API* to generate and post the topologies to the *Aurora scheduler*, a service that runs on top of Mesos. Initially, the topologies are forwarded to *Heron* and the *heron client* interacts with Aurora and deploys all the required components to handle the topologies. *Heron* runs on Aurora with the cluster set up of Zookeeper and required configurations of Heron to talk with Zookeeper. Heron has an abstraction that enables to run on different schedulers like Mesos, Amazon EC2 [17] and Mesos. However, in Storm, the component *Nimbus* serve as a scheduler. Every topology will act as an *Aurora job* that constitutes multiple containers, where each container starts a *heron-executor process* that executes the *Topology Master* or *Heron Instances* (spout/bolt) that are allotted to the container [249]. The *Topology Master* always runs on container 0. The rest of the containers execute on a *Stream Manager*, a *Metrics Manager* and on multiple Heron instances. The *Stream Manager* governs the routing of tuples among topology units. A *Heron Instance* is a unique task of spout/bolt that enables debugging and profiling. A *Heron Instance* (HI) of a topology associates with its local *Stream Manager* and all of the other Stream Managers to form a network by linking to each other. A Metrics Manager (MM) in a topology accumulates and scatters metrics from distinct components in a container. Also, MM sends the metrics to both TM and external accumulators (e.g., Scribe, Graphite). The current version 0.17.5 of *Heron* ensures the exactly once delivery mechanism.

4.4.2 Cloud-based streaming solutions

Cloud-based streaming systems are proficient in processing streaming data for various applications. The major advantage of a streaming analytics engine is the ability to facilitate real-time analytics and produce output, so that the data becomes actionable with the least possible delay. Amazon Kinesis, Azure Stream analytics, and IBM stream analytics play a significant role in streaming analytics and provide a convenient mechanism to work with other services of respected organizations.

– *Amazon Kinesis* is a cloud-based solution for processing streaming data that allows us to develop custom applications and process or analyze data streams for specific requirements [64]. Kinesis captures data continuously and stores terabytes of data per hour from various sources such as business transactions, web clickstreams, location-based tracking events, social network feeds. Figure 13 illustrates the architecture of the Kinesis data stream. A *data stream* in kinesis is a set of shards. Each shard comprises a sequence of data records. A shard consists of a fixed data capacity. A data record is a data unit that is

[17] https://aws.amazon.com/ec2/.

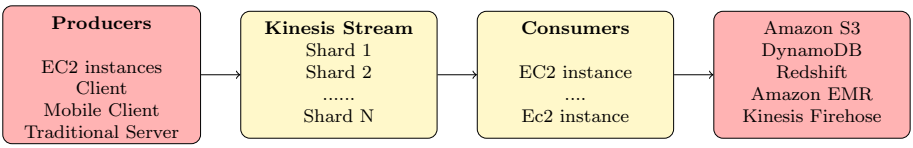

Fig. 13 Kinesis data streams high-level architecture

Fig. 14 Core features of Azure Stream Analytics

stored in a Kinesis data stream. Kinesis data streams assign a sequence number to each of the records. The elements of the data records are *sequence number*, *partition key* and *data blob* (maximum size is 1MB) [253]. The *partition key* does the assignment of a data record to a particular shard in the stream. A *sequence number* is a unique identifier of the record within its shard. A data blob is the data of interest that a producer adds to a Kinesis data stream. Producers send the records to kinesis data stream. A producer may be a client or a mobile client or a web server that sends log data to a stream. Consumers receive the records from kinesis data streams and process them. The consumers can also be called as Kinesis Data Stream applications. A consumer of a stream is a Kinesis Data Stream application that usually executes on a set of EC2 instances. An Amazon Kinesis Data Stream application can be developed by employing Kinesis Client Library (KCL) or Kinesis Data Streams API (KDS). The output of KDS application may consider for the input of another stream that enables us to design complex topologies to process the real-time data. Also, an application can forward data to various AWS services like DynamoDB, Redshift, EMR and Firehose. The Kinesis *Data Firehose* [18] provides the easiest mechanism to capture, transform and load the streaming data into Amazon S3, Redshift, Amazon Elasticsearch service empowering near real-time processing with available BI (Business Intelligence) tools.

– *Azure Stream Analytics*, an efficient event-processing engine configured for real-time analytic processing on streaming data. The data will generate from diverse sources, sensors, social media feeds, web clickstreams, infrastructure systems, etc. Stream analytics works efficiently to investigate massive volume of data from a variety of devices, processes and obtain information from data streams, recognize patterns, trends and correlations. The obtained patterns are used to trigger other actions like alerts, feeding data to reporting tools or save it for further study [254]. Figure 14 illustrates the Streaming Analytics pipeline, depicting the steps involved are data ingestion, analyzation and then forward for presentation or action.

  The first phase in Stream Analytics begins with acquiring the streaming data from a source. In the next phase, an event hub or IOT hub ingest the data into Azure from a device or Azure Blob Storage. To explore the stream, build a Stream Analytics job that describes the source of data. The job also specifies transformations like Create, Delete, Get and Update. These transformations are used to identify data, patterns and relationships.

[18] https://aws.amazon.com/kinesis/data-firehose/.

In identifying the patterns/relationships, Stream Analytics governs SQL-like language to filter, sort, aggregate and join data streams during a particular period.

– *IBM Streaming Analytics (SA)*, an advanced analytic platform for IBM Cloud that can operate to ingest, analyze and identifying correlation patterns as it comes from diverse sources in real-time [255]. IBM SA analyzes a broad range of data including unstructured text, audio, video, geospatial data in real-time to get potential patterns. Using IBM Watson,[19] the streaming data can be converted from speech to text. The *speech-to-text* tool allows users to build applications that extract voice, transform it to text by performing natural language processing tasks.

4.4.3 Comparison of distributed streaming systems

In this section, we summarize the recently developed stream processing systems including cloud-based streaming systems. We compare all the streaming systems in Table 9 concern to data abstraction, delivery mechanism, fault-tolerance, memory management, latency, throughput, and limitations.

– *Samza and Storm*
The differences between Samza and Storm are: i) Samza employs single-thread processes (container), where each container consists of many tasks. However, only a unique thread invokes all the different tasks. In the sense that, every container maps only one CPU core which minimizes the resource utilization and involvement of other tasks processing on a particular machine [256]. Storm uses a multi-threaded model, where it can handle the excess volume at less cost of resource utilization of a machine. ii) Another convenient feature in Storm is dynamic rebalancing which is useful to add of multiple processes to a topology without starting the topology or cluster again and again. Samza does not consist of dynamic rebalancing feature. iii) Currently, Samza 0.14.0 has only at least once delivery, whereas storm has at most once and at least once delivery mechanisms. However, Storm offers exactly once semantics delivery mechanism using Trident abstraction [257].
– *S2 and Samza*
S2 divides the data into a sequence of micro-batches for a fixed time (one second), where each batch data can be characterized as an RDD. A continuous (unbounded) sequence of RDDs can be called as a DStream in S2. Unlike S2, Samza processes the messages in the order they arrive in the stream partition. Samza achieves parallelism by dividing the processing of messages into individual tasks in a parallel approach. However, depending on the workload and latency requirement one can run multiple tasks in a single container or run one task per container. The parallelism in S2 can be maintained by dividing the job into small batches and forward them to executors. S2 has two types of parallelism. First one is parallelism while receiving the streams and the second is, the parallelism in processing the streams. S2 does not provide the guarantee of at least once or at most once delivery semantics due to the loss of data at some conditions, where the driver program fails.
– *Apache Flink, S2 and Samza*
Flink processes both batch and streaming data. Flink computes batch programs as a special kind of streaming programs, in which the streams are finite (bounded). S2 do iterative computing in micro-batches, whereas Flink can handle the iterative tasks utilizing its streaming architecture. Similar to Samza, the Flink tasks run on a cluster, and

[19] https://www.ibm.com/watson/.

the JobManager handles the checkpoint coordination. In the same point of view, Samza can use exactly once semantics like the similar mechanism in Flink. A major distinction between Samza and Flink is that Samza has the shared channel problem (SCP) [242], whereas Flink does not have.

– *Storm and S2*
S2 employs micro-batch processing, whereas Storm uses per-tuple processing, with consequences trade-offs between latency and throughput [258].
– *MillWheel and other systems*
In achieving fault-tolerance, MillWheel checkpoints its improvement at fine granularity and eliminates the requirements to buffer incomplete data at external senders for long time intervals between checkpoints [246]. S2 do well for the effective checkpointing, but it restricts the operator's space that is accessible from user code. The Storm itself does not provide some of the delivery semantics. But Storm provides exactly once delivery semantics with Trident, an interface which built on top of Storm and ensures persistent state [259]. However, the limitation of Google MillWheel is, it performs poorly for problems that are difficult to parallelize between distinct keys.

Table 9 illustrates several stream processing systems in respect of the parameters data abstraction, message delivery, fault-tolerance, memory management, latency, throughput and limitations. We observe the following characteristics by comparing all the streaming tools relevant to the parameters.

– The data abstractions in Storm, Samza, S4, S2, Flink, SAMOA, MillWheel and Heron are Topology (Spout, bolt), Partitioned streams, Processing elements (PEs), DStreams, Data Streams, Topologies (Processor, Content event), user-specified transformations (computations) and Topologies (DAG) respectively.
– Except for S4, all the streaming tools in Table 9 provide *exactly once* delivery semantics. The Storm achieves fault-tolerance using the daemons, *Nimbus* and the *Supervisors*. These daemons are designed to be stateless and fail-safe. Apache Samza uses checkpointing method to maintain fault-tolerance. However, S4 is partially fault-tolerant, because the state of the processes saved in memory is lost in the time of handoff. S2 achieves fault-tolerance using checkpointing and the concept of *Journaling* (*write-ahead-logs*). Flink accomplishes the fault-tolerance by using Chandy–Lamport and lightweight distributed snapshot mechanism. Moreover, the tools MillWheel and Heron maintains fault-tolerance using checkpointing and continuous checking of the stateful scheduler
– All the streaming tools except S2 achieve low latency and high throughput. But, S2 provides high latency and lower throughput due to its in-memory and immutable nature of RDDs.
– In spark, when the data objects are too large, the simplest way to decrease memory usage is to save the data objects in a serialized form using the *StorageLevels* in the RDD persistence API. *Execution* and *storage* are the two categories of Spark in which the memory usage can fall. Execution memory denotes the memory which is used for computations (joins, shuffles, etc.) and storage memory is used for caching and internal communication across the cluster. Spark.executor.memory is meant for the amount of memory to use per executor process. Flink enables a hybrid approach that combines on-and-off-heap memory allocations.
– The limitation of S4 is, its memory management mechanism is very simple, but not the efficient system. S2 streaming mechanism is not suitable for real-time computing. The limitations of Flink are inefficiency in resource utilization, Chandy–Lamport snapshot (checkpointing) mechanism identifies the data at a homogeneous point of view of states

Table 9 Comparison of distributed stream processing systems

Framework	Data abstraction	Message delivery/ semantics	Fault-tolerance	Memory management	Latency	Throughput	Limitations
Storm [224] (Real-time processing)	Continuous Operator Model. Topology, Spout, bolt	At most once At least once Exactly once (Stateless)	Nimbus and the supervisors	Through JVM	Low	High	Not applicable for batch processing
Apache Samza [229]	Partitioned Stream model	At least once embedded key-value store	Checkpointing	Using stateful stream processing	Low	High	Does not support exactly once delivery
S4 [231]	Actors programming paradigm. Processing Elements (PEs)	Does not support exactly once semantics	Partially fault-tolerant. Uncoordinated checkpointing	By assigning a time-to-live (TTL) to every PE object	Low	Maximum	Memory management system is simple, but not the most efficient
Spark [209,250] Streaming S2 (Batch & Stream processing)	Discretized Stream (DStream) using RDDs	Exactly once	Write ahead logs (Journal), Metadata checkpointing, Data checkpointing	Serialized persistence of RDDS. Total memory of an executor is assigned to the Java heap (spark.executor.memory)	High	Lower	Divides the data into batches and not applicable for real-time processing. No file management system. No guarantee for at least once or at most once message delivery semantics

Table 9 continued

Framework	Data abstraction	Message delivery/ semantics	Fault-tolerance	Memory management	Latency	Throughput	Limitations
Apache Flink [251] (Batch and Stream processing)	DataSet DataStream	Exactly once	Chandy-Lamport distributed snapshots. light-weight snapshot	Custom memory management. Flink enables a hybrid approach, combining on- and off-heap memory allocations. Flink serializes data into memory segments instead of using JVM heap to avoid long running JVM garbage collector stalls	Low	High	Inefficiency in resource utilization.
Apache SAMOA [243]	Basic topologies—Processor, Content Event, Stream, Task, Topology Builder, Learner, Processing Item	Exactly once	fault-tolerant	Works with Storm, Samza and S4	Low	High	Only execute on a single machine, but does not scale into multiple machines

Table 9 continued

Framework	Data abstraction	Message delivery/ semantics	Fault-tolerance	Memory management	Latency	Throughput	Limitations
Google [246] MillWheel	User-specified, directed graph of computations	Exactly once	Checkpointing	In-memory caches or aggregates.	Low	High	Poorly suited for monolithic operations. Does not perform well for problems that are difficult to have parallelism between distinct keys.
Twitter Heron [247,252] (Real-time processing)	Modules communicate via basic inter/intra process communication mechanisms	At most once at least once Exactly once	The stateful scheduler continuously checks and takes required actions to recover from failures	The Resource Manager module takes care about the utilization of resources allocated for a specific topology	Lower latency than storm	High	N/A

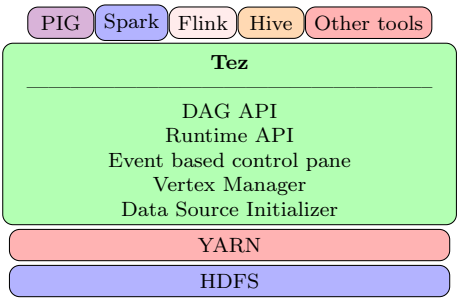

Fig. 15 Components of Tez

for complex tasks. However, SAMOA executes only on a single machine but not scale to multiple machines. Storm is suitable for real-time computing only, but it does not handle batch processing jobs.

4.5 Interactive analytical processing tools

Interactive analytical processing tools are another category of big data tools that address the requirements of interacting various tools to process large-scale datasets of various applications. Interactive analytical processing tools provide an interactive environment where the users can access the data and make an analysis of data.

- *Apache Tez* provides a unifying framework to construct scalable and efficient data-flow-centric engines. Apache Tez is useful to create purpose-built engines that allow data processing according to their particular requirements. Tez enables users to represent computation in the form of directed acyclic graphs. Figure 15 shows the components of Tez architecture.
  DAG (Directed Acyclic Graph) API, Runtime API, event-based control pane, Vertex manager and Data Source Initializer are the key components in building data-flow-driven processing runtimes [260]. The role of these APIs is to govern frameworks to model the physical and logical semantics for data flow graphs with less code. Tez is not an engine, rather than it consists of a library of components which are used to construct data-flow-based engines. The performance of Tez-based integration of Spark, PIG, Hive on YARN is much better than their native YARN execution on standard benchmarks TPC-H and TPC-DS [261].
- *Hive* was initially developed by Facebook and adopted by many other companies like Amazon, Hortonworks [212,262]. It is a data warehouse solution that represents queries in a declarative language called as HiveQL with similar features of SQL. The major tasks of Hive are, conversion of HiveQL into MapReduce jobs and execution of that jobs using Hadoop [212]. The purpose of Hive is to handle with ad hoc querying, data summarization and big data analysis. It consists of a system catalog, driver, query compiler, execution engine and Hive server (JDBC/ODBC driver), a metastore consists of schemas for query optimization. Hive eases the process of integration between Hadoop and business intelligence/visualization tools. Hive uses MapReduce/Tez frameworks for scheduling, data mobility and parallelization to process SQL queries.
- *Impala* is an open-source tool for ad hoc querying using SQL in *HE* [213]. Unlike Hive, Impala has a shared-nothing architecture and running its own daemons on each node of the cluster. Impala uses an MPP engine which is an alternative to MapReduce. With

Impala, the data can query quickly from HDFS or HBase tables that improves Hadoop capability to refine several kinds of data types and provides schema at runtime. It carries input data information that includes data types, partition names using Hive metastore. The computing potential of Impala is high due to its I/O subsystem. The major components of Impala are query planner, coordinator and execution engine. The difference between Impala and Hive is, Impala executes SQL queries using MPP engine, whereas Hive implements SQL queries using MapReduce.

4.6 Container orchestration platforms

In this section, we discuss the different Container Orchestration platforms. Containers are designed to provide a method of virtualization that enables an application and its dependencies to run in the resource-isolated environment. Resource-isolation means, the failure of a single container due to excessive resource utilization by a process, it affects only that particular container but does not affect on the entire server [263]. The three main properties in containers are *isolation*, *orchestration* and *scheduling* [211]. In *isolation*, the failure of one computing unit cannot affect the other unit. Allocating resources in a balanced manner geographically to distribute load is known as orchestration. Scheduling takes the responsibility of detecting and replacing the failures immediately. Some of the recently developed orchestration platforms for the availability and cluster management are Kubernetes [210], Mesos [264] and Amazon EC2 Docker.[20]

– *Kubernetes*
Kubernetes developed by Google, that supports a container-based, an open-source system for handling containerized applications over multiple hosts, and provides fundamental mechanisms for deployment, governing and scaling of applications [210]. On behalf of user workloads, Kubernetes organizes storage infrastructure, computing and networking. Kubernetes has multiple features and can be thought of as a container platform (Platform as a Service-PaaS), a microservices platform and a portable cloud platform. The objective is to support a wide variety of workloads including stateless/stateful and preprocessing tasks. Kubernetes is not a conventional all-inclusive platform, instead it works at the container and the hardware level. It has some commonly applicable PaaS features like deployment, scaling, load balancing, logging and monitoring. The fundamental working unit in Kubernetes is a *pod*, an abstraction of a collection of containers fixed with some shared resources. The key components in Kubernetes are *pods* and *kubelet agent* [265]. *Pods* are groups of one or more containers that share location, scheduling, namespaces and volumes that run on a host machine. The *kubelet agent* governs these *pods* and serves as a health check monitor interface with the Docker daemon. If any component in *kubelet agent* is not working, it has the ability to restart that component. Kubernetes works on master-slave perspective and supports a cluster set up with master and worker nodes. In Kubernetes a slave can also be called as a minion. The correspondence between the Kubernetes master and a worker (slave) can be accomplished through the *kubelet* service [266]. The execution of *kubelet* can occur on each machine in the cluster. A node that acts as a master can also accomplish the task of a slave during implementation. Since the Kubernetes works with Docker containers, the docker daemon will run on every node in the cluster.

[20] https://aws.amazon.com/docker/.

– *Mesos*
A platform for data center computing. Mesos is a distributed kernel, a meta-scheduler or scheduler of schedulers, which uses a container architecture that allows the execution of multiple distributed systems [264]. The objective of Mesos is to construct a scalable and well-organized system that coordinates a broad range of frameworks. Mesos consists of a two-level distributed scheduling mechanism known as resource offers. The mechanism takes in two steps. First one is to determine the number of resources required for each framework and the second job is, what kind of resources to accept by each framework, and which type of computation to run on each framework. Mesos works on a master–slave process. The master process governs the slave daemons working on every cluster node and the frameworks run the jobs on the slaves. The purpose of Mesos is to share commodity clusters among various computing frameworks like Hadoop and Message Passing Interface (MPI) [267].
– *Amazon Elastic Compute Cloud (Amazon EC2) Docker* is a software platform that enables to build, test and deploy distinct applications [268]. The Docker combines software into standard units called *containers* that constitute required software to execute the code, libraries, system tools and the runtime. A user can rapidly set up and scale applications into a compatible, required environment and can observe the code that is running. Docker runs on Amazon Web Services (AWS) and provides a highly reliable, low-cost services to build, transport and runs applications in a distributed approach at any scale.

4.7 Distributed message queueing systems

Messaging systems are essential in sending the data from a source to a destination based on the publish/subscribe paradigm. A Message broker is an intermediary program that provides a common platform to handle message validation, transformation, and message routing tasks while performing the communication between the components of the distributed systems. Apache Kafka [269], RabbitMQ [270], Apache ActiveMQ [271] are the widely deployed open source message brokers. However, AmazonMQ is a cloud-based managed message broker service for ActiveMQ developed at Amazon Web Services [272]. We now briefly discuss Kafka and RabbitMQ message brokers.

– *Kafka*, an efficient tool for handling a huge volume of log data processing and developed at LinkedIn in 2010 [269]. It is a distributed log service which achieves higher throughput than conventional messaging systems. Figure 16 illustrates the architecture of Kafka. The core concepts of Kafka are *Topic*, *Producer*, *Broker*, and *Consumer*. A *Topic* consists of a stream of messages of a particular type. The role of the *Producer* is to publish messages to a Topic. Then the published message is distributed to the subscribers of the Topic. *Brokers* are the servers at which the published messages are stored. A *Consumer* pulls the data from the brokers and consumes the subscribed messages of Topics. Each Topic consists of a partitioned log in which a set of published messages are stored for a specific time span. Kafka cluster maintains replication of messages for the partitions and distributes to existing servers for fault-tolerance [273]. Kafka comprises a distributed log, which consists of a sequence of published and consumed messages for a particular Topic. Kafka has a built-in tool known as Kafka connect which persistently send the data into Kafka or send the data into external systems.
Kafka works on publish–subscribe perspective. kafka has a significant feature in which it recollects the information about messages for a specific duration of time. Although a consumer ingests information, the Kafka cluster maintains that message for a particular

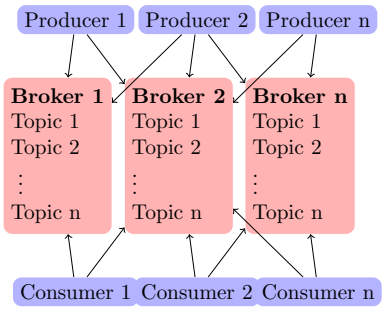

Fig. 16 Kafka architecture

time interval. Kafka has another feature of abstraction known as consumer community. Kafka follows the queuing system when the total consumers subscribe a Topic in the same community. In that case, each message is carried to only one consumer. Also, if each consumer subscribes to a Topic of the distinct community, Kafka works based on the publish–subscribe system in which every message is transmitted to respected consumers.

– RabbitMQ is a scalable messaging system, initially developed as an implementation of the Advanced Messaging Queuing Protocol (AMQP). The AMQP adopts a stream-based message transfer and carries sequential frames across multiple channels [274]. Each independent frame consists of a respected channel number and sizes of frames to enable receiver can efficiently describe them.

4.8 Data ingestion tools

The process of collecting and importing data into a data store/database is known as *data ingestion*. An effective data ingestion process starts by harmonizing data sources, authorizing every document and routing information to the exact destination [275]. Flume [276], Sqoop [277], and Chukwa [278] are some widely used data ingestion tools which play a vital role in transferring the data from databases to Hadoop.

– *Sqoop* transfers huge volumes of structured data between databases and Hadoop [277]. Sqoop enables import and export of data from external sources to HDFS. It works with Oozie to schedule and automate import and export operations. The import process in Sqoop occurs in two stages. In the first stage, it searches the database to acquire the required metadata for the data being transferred. In the second stage, Sqoop submits the map-only job to the cluster, which actually does the data transfer using the metadata collected in the first phase [279].
– *Flume* provides a distributed, reliable service that acquires log data from different applications and delivers into Hadoop [276]. It allows Hadoop users to collect, aggregate and deliver log data into HDFS. An event in Flume is a data flow unit that consists of a byte payload and a set of strings. A Flume agent is a Java Virtual Process (JVM) through which events flow from an external source to a sink. Flume receives an event from its source like web server and stores it into one or more channels. Then the sink receives the event from the channel and sends it into HDFS or sends the event to the source of next Flume agent. Figure 17 illustrates the data flow in Flume.
– *Apache Chukwa* [278] is an efficient tool built on top of Hadoop for supervising and analyzing results for efficient use of collected data.

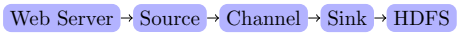

Fig. 17 Data flow in Apache Flume

4.9 Large-scale graph processing systems

In this section, we explain the recently developed large-scale distributed graph processing tools. The new directions for the development of graph processing systems for large-scale networks are increasing due to the rapid growth of network data. The data is flowing in huge volumes from diverging sources of complex networks. Even though we have several existing graph processing frameworks such as Pregel [44], GraphLab [280], Giraph [281], PowerGraph [282] and GPS [283], we present the most recently developed distributed graph processing tools namely Graphx and Gelly with their core features.

– *GraphX* A distributed graph processing tool built on top of Apache Spark [181,284]. The *General-purpose parallel systems* or *data-parallel systems* like Hadoop MapReduce are not well suited to build graph-based operations from unstructured datasets. This is due to building graph applications involve multiple iterative tasks. *Data-parallel systems* are suitable for the independent computing tasks, whereas *graph-parallel systems* can apply for dependent (iterative) tasks like node neighborhood computation using vertex and edge properties of a graph [285]. The drawbacks of *graph-parallel systems* are, i) They are not suitable to construct the graphs from diverse data sources, ii) unable to compare distinct graph structures, and iii) coarsening the graph. To overcome these limitations, GraphX combines both *data-parallel* and *graph-parallel* characteristics by having a new graph data model that consists of Property graph, collection operators, Graph operators, and Composing operator features that are shown in Fig. 18. A *Property graph* is a representation of graph structure which combines the user-specified function with vertex and edge properties. The collection operators (map, filter, reduceByKey) takes collection and output another collection by the removal, aggregation or transformation of the attributes. The operator leftJoin accepts input as vertex collection, partition the graph, join the vertices and produce a new collection of vertices by preserving the initial graph structure. A triplet comprises the attributes of an edge with its two vertices. The mrTriplet operator applies the map user-defined function on every triplet, sends the messages to the source and destination vertices. GraphX partitions both the edges and vertices using hash-partitioning. Vertices are partitioned in respect of vertex_ids, i.e., the vertices are grouped by a hash index in every partition. Every vertex partition comprises a *routing table* and a *bitmask*. The *bitmask* consists of vertex information and allows the operations intersection/filtering used by subgraph/join operators. In a partition, the edges are clustered (grouped) with respect to source vertex_Id using compressed sparse row (CSR) and hash-indexing techniques.
The ungrouped index exists on destination vertex_Id. The CSR technique is a mapping between source vertex_Id and the corresponding block of outgoing edges. The *routing table* maps a vertex_Id to a collection of partitions with neighboring edges. GraphX ensures the composition operators like coarsening which merges nodes of a graph to obtain the next coarser of the graph. The data abstraction in GraphX constitutes VertexRDD, EdgeRDD, EdgeTriplet, and Graph to represent the vertices, edges of a graph using the concept of property graph. GraphX consists of modified Pregel API. Instead of sending messages to all nodes in Pregel, GraphX sends messages to only adjacent vertices for building parallel messages using a user-specified message function [286].

GraphX version 2.2 includes the PageRank, Triangle counting and Connected components algorithms.

– *Gelly* A distributed graph processing tool (Graph API) for graph analytics built on top of Apache Flink [146]. In Gelly, a graph can be denoted as a Vertices DataSet and Edges DataSet. A vertex in a graph consists of vertex_Id, vertex_value. An edge in a graph comprises of source_Id, target_Id and edge_value. The Graph class in Gelly embedded with the methods transformations, neighborhood aggregations, and general graph metrics. Transformations are the methods to allow distinct graph operations to apply on vertices, edges, and triplets of a particular graph to get another modified graph. Neighborhood aggregations permit vertices to compute aggregation on neighborhoods of adjacent vertices. The general graph metrics include characteristics of graphs like vertex degree and number of vertices/edges. Figure 19 illustrates the components of Gelly with various graph characteristics. Gelly adopts the *vertex-centric* programming model incorporated into distributed framework Apache Flink. The *vertex-centric* model iteratively computes a user-specified program over vertices of a graph.
The vertex program works in the perspective of a vertex, receives input as the vertices information and neighboring vertices information along incident edges. Also, Gelly adopts *Scatter-Gather*, another iterative processing model. In Scatter-Gather, the task Scatter sends the value of a vertex to its adjacent nodes and the task Gather receives the messages and updates the vertex value. Moreover, Gelly supports *Gather-Sum-Apply* (GSA) model developed by Powergraph [282]. In contrast to *vertex-centric* model, GSA model pulls the node information from its adjacent vertices and pushes the updates from the incoming neighbors [183]. *Gather* is a user-specified function that invokes edges and adjacent vertices of each vertex in parallel. The phase *Sum* aggregates the values collected from gather phase into a single value using a user-specified Reducer. The method *Apply* updates the value of each vertex by applying a function to the present value and the aggregated value provided by the sum phase. Gelly constitutes a set of scalable graph generators (path graph, star graph, grid graph, etc.), where each generator is scale-free and parallelizable. The execution Gelly provides methods to built, convert, and alter graphs along with a collection of graph algorithms. The current version 1.4 of Gelly library consists of the algorithms on similarity metrics (Adamic-Adar, Jaccard index), Hyperlink-Induced Topic Search (HITS) and PageRank that are significant for link-based problems. Moreover, Gelly also has a Community detection algorithm using Label Propagation strategy. Furthermore, the library includes clustering coefficient (local, global, and average), GSA-Single Source Shortest Paths (G-SSSP), GSA-Connected Components, and Triangle Enumerator algorithms. Currently, the Researchers are working to enhance Gellys DataStream to build its streaming engine to handle the graph-based streaming applications
– *GRADOOP*, an open-source, scalable-graph data flow system for declarative analytics of heterogeneous graph data. Gradoop built on top of Flink for scalable graph analytics [287]

Other tools

– *Zookeeper* is a service that enables highly reliable coordination of processes for distributed applications [288]. Zookeeper provides a high-performance coordinating kernel to build critical coordination primitives for the client requests. Zookeeper maintains an API that employs a wait-free data objects

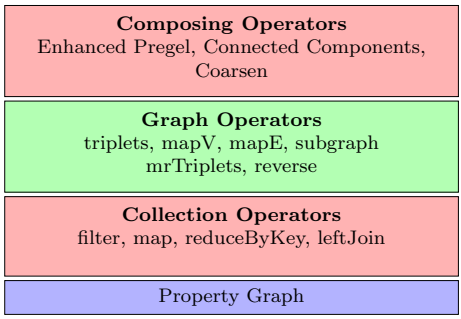

Fig. 18 Data model of GraphX

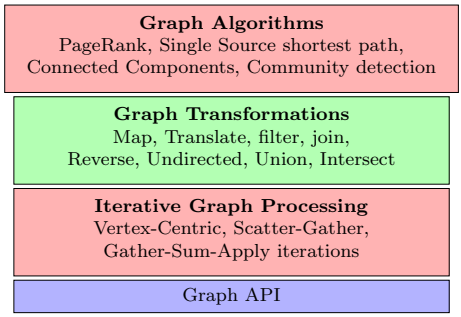

Fig. 19 Gelly core components

- *MYRIAD* The aim of the Apache Myriad project is to integrate YARN with Mesos [289]. Apache Myriad allows co-occurrence of Hadoop and Mesos on the same physical hardware. It is used for scaling YARN clusters dynamically that allows running Hadoop applications like Spark alongside, non-Hadoop applications such as Node.js, Memcached, RubyonRails (ROR). YARN works as a Mesos framework, with resource and node managers that run inside Mesos repositories.
- *Apache Avro* is a data serialization framework that converts data from various formats into a binary form for processing [290].
- *Apache Falcon* A framework for managing data lifecycle in Hadoop clusters. It allows processing a single large dataset in HDFS in several ways for batch, interactive, and streaming applications [291].
- *Nifi* is a reliable and powerful Web-based user interface tool to distribute and process the data [292].
- *Oozie* a workflow scheduler system to monitor Hadoop jobs [293].
- *Apache Ignite* is a distributed in-memory platform for processing and managing massive datasets in real-time with high-performance.

Summary

In this section, the major technologies, tools that are essential to analyze large-scale datasets are presented. The big data tools can be categorized as batch processing tools, stream processing tools, and interactive analytical processing tools. We discussed the recently developed improvements in Hadoop 3.0 especially the concepts of journaling and HDFS HA features which are necessary to maintain fault-tolerance. Also, We presented the current developments in Spark 2.3 and focused on the concepts of structured streaming and Spark works by inte-

grating with Kubernetes. Since querying the data from the Hadoop environment has much significance, we compared several SQL querying tools on Hadoop such as Hive, Impala, SparkSQL, Presto, Apache drill, Kylin, Vertica, Google Tenzing and Hadapt. We compared these SQL tools on 10 parameters and drawn some observations. Moreover, we discussed distributed streaming systems (Storm, Samza, S4, S2, Flink, SAMOA, MillWheel, Heron) and cloud-based streaming solutions (Amazon Kinesis, Microsoft Azure stream analytics, IBM streaming analytics). We compared all the distributed streaming tools on seven parameters including limitations. Also, we differentiated each pair of streaming tools emphasizing some strengths and shortcomings. Furthermore, we presented interactive analytical tools like Tez, Impala and Hive. Finally, we elaborated the core features of recently developed large-scale graph processing systems namely GraphX and Gelly.

5 Research directions and outlook

In this section, we briefly discuss the Research directions and opportunities according to the current trend of ML and big data analytics.

5.1 Complex networks and big data

At present *Complex network data* from social networks, terrorist networks, transportation networks, IOT network, PPI (protein–protein interaction) networks, etc., is growing rapidly and one of the major research area that has many opportunities to develop ML techniques in complex networks for big data to find potential patterns. In analyzing large-scale networks, there is a necessity to develop advanced graph analytics such as Structural similarity measures, characterization of structural properties of networks which remain invariant for dynamic networks, cascading behavior of nodes, distributed computation for SimRank and detecting abnormal nodes with new processing frameworks (Spark, Flink) are some of the open issues.

5.2 Machine learning and big data

We notice that a lot of research problems are open in big data analytics, especially in the domain of developing parallel and distributed ML algorithms for classification, clustering, detecting communities in large-scale networks, frequent subgraph mining and web-based graph mining. Parallel and distributed approaches for finding frequent itemsets using N-lists, Nodesets and diffsets are open problems in the area of frequent pattern mining. Moreover, mining closed and maximal frequent itemsets in distributed approaches is one of the open problems in developing ML algorithms. Mining communities in a large network using parallel and distributed approach are still at a growing stage. Even though Spark MLlib and Flink ML has some fundamental and standard ML algorithms, but they are still at a growing stage and more advanced ML algorithms have to develop which create plenty of opportunities in the area of agriculture, bioinformatics, educare, financial services, healthcare, real-time governance, power sector, and scientific research. In streaming systems, there is a scope to develop online ML algorithms that are used for better search and process Real-time data to provide faster solutions to the end users.

6 Summary and conclusions

The objective of this survey is to a provide a comprehensive overview of the analysis of massive datasets by focusing on the advanced tools with recent developments, which are essential to load, store, manage, process and visualize the large-scale datasets. The knowledge of analyzing of large-scale structured and unstructured datasets is essential and has a great significance in the trend of big data. In this paper, we have explored the big data system, technical aspects and the components in each stage in a generalized perspective. We have investigated and classified existing big data frameworks, tools and technologies from different aspects. First, we have presented key stages including data sources, data management, computing frameworks and data analysis in a generalized view. The main components of each stage, to process the massive datasets with recent developments in a comparative approach are described. Second, comparison of several distributed file systems and MapReduce-supported NoSQL databases were presented in our work. NoSQL databases play a crucial role in the data management process for refining big data. The above comparisons are useful to distinguish the performance of various distributed file systems and effective querying of unstructured data from NoSQL databases. Also, we classified the analytics concerning nature of the data, domain type and type of application. Furthermore, we compare several distributed and cloud-based ML tools by investigating some case studies. Also, various visualization tools are presented with their strengths and drawbacks. Finally, we presented batch processing, distributed/cloud-based stream processing tools, interactive querying tools, container orchestration tools, distributed messaging systems and data ingestion tools. The comparison of several SQL query tools on Hadoop while specifying the significance of Hadoop 3.0 Ecosystem integrated with distinct data processing tools with recent developments were also presented in our work. These comparisons are helpful in utilizing the suitable tools properly for processing various big data applications.

References

1. The size of the world wide web (the internet). http://worldwidewebsize.com/
2. Mattmann CA (2013) Computing: a vision for data science. Nature 493(7433):473–475
3. National Aeronautics and Space Administration. https://www.nasa.gov/
4. Clavin W (2013) Managing the deluge of 'big data' from space. NASA Jet Propulsion Labratory
5. Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. Comput Netw 54(15):2787–2805
6. SCB Intelligence (2008) Six technologies with potential impacts on us interests out to 2025. National Intelligent Concil, Tech. Rep
7. Yu S, Liu M, Dou W, Liu X, Zhou S (2017) Networking for big data: a survey. IEEE Commun Surv Tutor 19(1):531–549
8. Pouyanfar S, Yang Y, Chen S-C, Shyu M-L, Iyengar SS (2018) Multimedia big data analytics: a survey. ACM Comput Surv 51(1):10
9. Alaba FA, Othman M, Hashem IAT, Alotaibi F (2017) Internet of things security: a survey. J Netw Comput Appl 88:10–28
10. Zikopoulos P, Eaton C, et al (2011) Understanding big data: analytics for enterprise class hadoop and streaming data. ISBN: 0071790535
11. Chen M, Mao S, Liu Y (2014) Big data: a survey. Mob Netw Appl 19(2):171–209
12. Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU (2015) The rise of big data on cloud computing: review and open research issues. Inf Syst 47:98–115
13. Ma C, Zhang HH, Wang X (2014) Machine learning for big data analytics in plants. Trends Plant Sci 19(12):798–808
14. Laney D (2013) 3d data management: controlling data volume, velocity and variety. META Group Research Note 6(70), 1
15. Fan W, Bifet A (2013) Mining big data: current status, and forecast to the future. ACM sIGKDD Explor Newsl 14(2):1–5

16. Demchenko Y, De Laat C, Membrey P (2014) Defining architecture components of the big data ecosystem. In: Collaboration technologies and systems (CTS), 2014 international conference on, pp 104–112
17. Fernández A, del Río S, López V, Bawakid A, del Jesus MJ, Benítez JM, Herrera F (2014) Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. Wiley Interdiscip Rev: Data Min Knowl Discov 4(5):380–409
18. Assunção MD, Calheiros RN, Bianchi S, Netto MAS, Buyya R (2015) Big data computing and clouds: trends and future directions. J Parallel Distrib Comput 79:3–15
19. Emani CK, Cullot N, Nicolle C (2015) Understandable big data: a survey. Comput Sci Rev 17:70–81
20. Schuelke-Leech B-A, Barry B, Muratori M, Yurkovich BJ (2015) Big data issues and opportunities for electric utilities. Renew Sustain Energy Rev 52:937–947
21. O'Leary DE (2015) Big data and privacy: emerging issues. IEEE Intell Syst 30(6):92–96
22. Kune R, Konugurthi PK, Agarwal A, Chillarige RR, Buyya R (2016) The anatomy of big data computing. Softw: Pract Exp 46(1):79–105
23. Bello-Orgaz G, Jung JJ, Camacho D (2016) Social big data: recent achievements and new challenges. Inf Fusion 28:45–59
24. Bajaber F, Elshawi R, Batarfi O, Altalhi A, Barnawi A, Sakr S (2016) Big data 2.0 processing systems: taxonomy and open challenges. J Grid Comput 14(3):379–405
25. Nadal S, Herrero V, Romero O, Abell A, Franch X, Vansummeren S, Valerio D (2017) A software reference architecture for semantic-aware big data systems. Inf Softw Technol 90:75–92
26. Big data and veracity challenges. https://www.isical.ac.in/~acmsc/TMW2014/LVS.pdf
27. Gandomi A, Haider M (2015) Beyond the hype: big data concepts, methods, and analytics. Int J Inf Manag 35(2):137–144
28. Lee I (2017) Big data: dimensions, evolution, impacts, and challenges. Bus Horiz 60(3):293–303
29. Kung S-Y (2015) Visualization of big data. In: Cognitive informatics and cognitive computing (ICCI* CC), 2015 IEEE 14th international conference on, pp 447–448
30. Strohbach M, Ziekow H, Gazis V, Akiva N (2015) Towards a big data analytics framework for IoT and smart city applications. In: Modeling and processing for next-generation big-data technologies. pp 257–282. ISBN: 14-9783319385006
31. Wu X, Zhu X, Wu G-Q, Ding W (2014) Data mining with big data. IEEE Trans Knowl Data Eng 26(1):97–107
32. Wu X, Chen H, Wu G, Liu J, Zheng Q, He X, Zhou A, Zhao Z-Q, Wei B, Ming G (2015) Knowledge engineering with big data. IEEE Intell Syst 30(5):46–55
33. Wu X, Chen H, Liu J, Gongqing W, Ruqian L, Zheng N (2017) Knowledge engineering with big data (bigke): a 54-month, 45-million rmb, 15-institution national grand project. IEEE Access 5:12696–12701
34. Venner J, Wadkar S, Siddalingaiah M (2014) Pro apache hadoop. ISBN-13: 9781430248637
35. Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M (2009) A comparison of approaches to large-scale data analysis. In: Proceedings of the 2009 ACM SIGMOD international conference on management of data, pp 165–178
36. Teradata. http://www.teradata.com/Press-Releases/2016/Teradata-Announces-the-World%E2%80%99s-Most-Powerful
37. Chang L, Wang Z, Ma T, Jian L, Ma L, Goldshuv A, Lonergan L, Cohen J, Welton C, Sherry G et al (2014) HAWQ: a massively parallel processing SQL engine in hadoop. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, pp 1223–1234
38. Greenplum architecture. http://greenplum.org/gpdb-sandbox-tutorials/introduction-greenplum-database-architecture/
39. Ibm netezza. https://www-01.ibm.com/software/data/netezza/
40. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
41. Valiant LG (1990) A bridging model for parallel computation. Commun ACM 33(8):103–111
42. Lenharth A, Nguyen D, Pingali K (2016) Parallel graph analytics. Commun ACM 59(5):78–87
43. Apache hama project. https://hama.apache.org/
44. Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data, pp 135–146
45. Apache giraph project. http://giraph.apache.org/
46. Zhang H, Chen G, Ooi BC, Tan K-L, Zhang M (2015) In-memory big data management and processing: a survey. IEEE Trans Knowl Data Eng 27(7):1920–1948
47. Cai Q, Zhang H, Guo W, Chen G, Ooi BC, Tan K-L, Wong WF (2018) Memepic: towards a unified in-memory big data management system. IEEE Trans Big Data

48. Lim H, Han D, Andersen DG, Kaminsky M (2014) Mica: a holistic approach to fast in-memory key-value storage. USENIX, pp 429–444
49. Kuznetsov SD, Poskonin AV (2014) Nosql data management systems. Program Comput Softw 40(6):323–332
50. In-memory storage engine. https://docs.mongodb.com/manual/core/inmemory/
51. Chen CLP, Zhang C-Y (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf Sci 275:314–347
52. Mazón J-N, Lechtenbörger J, Trujillo J (2009) A survey on summarizability issues in multidimensional modeling. Data Knowl Eng 68(12):1452–1469
53. Hu H, Wen Y, Chua T-S, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. IEEE Access 2:652–687
54. Gantz J, Reinsel D (2011) Extracting value from chaos. IDC iview 1142:1–12
55. Kouzes RT, Anderson GA, Elbert ST, Gorton I, Gracio DK (2009) The changing paradigm of data-intensive computing. IEEE Comput 42(1):26–34
56. Labrinidis A, Jagadish HV (2012) Challenges and opportunities with big data. Proc VLDB Endow 5(12):2032–2033
57. UN Global Pulse (2012) Big data for development: challenges and opportunities. UN Global Pulse, New York
58. Kambatla K, Kollias G, Kumar V, Grama A (2014) Trends in big data analytics. J Parallel Distrib Comput 74(7):2561–2573
59. Chen Y, Qin X, Bian H, Chen J, Dong Z, Du X, Gao Y, Liu D, Lu J, Zhang H (2014) A study of SQL-on-hadoop systems. In: Workshop on big data benchmarks, performance optimization, and emerging hardware, pp 154–166
60. Mohammed EA, Far BH, Naugler C (2014) Applications of the mapreduce programming framework to clinical big data analysis: current landscape and future trends. BioData Min 7(1):1
61. Yang C, Huang Q, Li Z, Liu K, Hu F (2017) Big data and cloud computing: innovation opportunities and challenges. Int J Digit Earth 10(1):13–53
62. Oussous A, Benjelloun F-Z, Lahcen AA, Belfkih S (2017) Big data technologies: a survey. J King Saud Univ-Comput Inf Sci
63. Salloum S, Dautov R, Chen X, Peng PX, Huang JZ (2016) Big data analytics on apache spark. Int J Data Sci Anal, pp 1–20
64. de Assuncao MD, da Silva Veith A, Buyya R (2018) Distributed data stream processing and edge computing: a survey on resource elasticity and future directions. J Netw Comput Appl 103:1–17
65. Krumm J, Davies N, Narayanaswami C (2008) User-generated content. IEEE Pervasive Comput 4(7):10–11
66. White paper: How machine data supports gdpr compliance. https://www.splunk.com/pdfs/white-papers/splunk-how-machine-data-dupports-gdpr-compliance.pdf
67. Shameer K, Badgeley MA, Miotto R, Glicksberg BS, Morgan JW, Dudley JT (2016) Translational bioinformatics in the era of real-time biomedical, health care and wellness data streams. Briefings in Bioinformatics, bbv118
68. Marx V (2013) Biology: the big challenges of big data. Nature 498(7453):255–260
69. Cook CE, Bergman MT, Cochrane G, Apweiler R, Birney E (2017) The european bioinformatics institute in 2017: data coordination and integration. Nucleic Acids Res 46(D1):D21–D29
70. Akter S, Wamba SF (2016) Big data analytics in e-commerce: a systematic review and agenda for future research. Electron Mark 26(2):173–194
71. Aws: streaming data. https://aws.amazon.com/streaming-data/
72. Groenfeldt T, At nyse, the data deluge overwhelms traditional databases. https://www.forbes.com/sites/tomgroenfeldt/2013/02/14/at-nyse-the-data-deluge-overwhelms-traditional-databases/#25cda10f5aab
73. Sun J, Reddy CK (2013) Big data analytics for healthcare. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 1525–1525
74. Ranjan R, Georgakopoulos D, Wang L (2016) A note on software tools and technologies for delivering smart media-optimized big data applications in the cloud. Computing 98(1–2):1–5
75. Lloyd MD, Minor B. Harnessing the power of data in health. https://med.stanford.edu/content/dam/sm/sm-news/documents/StanfordMedicineHealthTrendsWhitePaper2017.pdf
76. Twitter statistics and facts. https://www.statista.com/topics/737/twitter/
77. Twitter by the numbers: stats, demographics and fun facts. https://www.omnicoreagency.com/twitter-statistics/
78. Number of monthly active facebook users worldwide as of 4th quarter 2017. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/
79. Rob Kitchin (2017) Big data. The International Encyclopedia of Geography

80. Gudivada VN, Baeza-Yates RA, Raghavan VV (2017) Big data: promises and problems. IEEE Comput 48(3):20–23
81. Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M (2015) Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun Surv Tutor 17(4):2347–2376
82. Raun J, Ahas R, Tiru M (2016) Measuring tourism destinations using mobile tracking data. Tour Manag 57:202–212
83. Kitchin R (2014) The data revolution: Big data, open data, data infrastructures and their consequences. Sage, ISBN: 13-9781446287484
84. Abiteboul S, Manolescu I, Rigaux P, Rousset M-C, Senellart P (2011) Web data management. Cambridge University Press, ISBN-13: 9781107012431
85. Ghemawat S, Gobioff H, Leung S-T (2003) The google file system. In: ACM SIGOPS operating systems review, vol 37, pp 29–43
86. Doctorow C (2008) Big data: welcome to the petacenre. Nat News 455(7209):16–21
87. Ovsiannikov M, Rus S, Reeves D, Sutter P, Rao S, Kelly J (2013) The quantcast file system. Proc VLDB Endow 6(11):1092–1101
88. Guerraoui R, Schiper A (1996) Fault-tolerance by replication in distributed systems. In: International conference on reliable software technologies, pp 38–57
89. Wiesmann M, Pedone F, Schiper A, Kemme B, Alonso G (2000) Understanding replication in databases and distributed systems. In: Distributed computing systems, 2000. Proceedings of 20th international conference on, pp 464–474
90. Shvachko K, Kuang H, Radia S, Chansler R (2010) The hadoop distributed file system. In: 2010 IEEE 26th symposium on mass storage systems and technologies (MSST), pp 1–10
91. Hdfs architecture. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
92. Schmuck FB, Haskin RL (2002) Gpfs: a shared-disk file system for large computing clusters. In: FAST, vol 2, pp 231–244
93. Jones T, Koniges AE, Yates RK (2000) Performance of the IBM general parallel file system. In: IPDPS, pp 673–681
94. Limitations: The IBM SONAS system. https://www.ibm.com/support/knowledgecenter/en/STAV45/com.ibm.sonas.doc/adm_limitations.h
95. Thanh TD, Mohan S, Choi E, Kim SB, Kim P (2008) A taxonomy and survey on distributed file systems. In: Networked computing and advanced information management, 2008. NCM'08. Fourth international conference on 1, pp 144–149
96. Beaver D, Kumar S, Li HC, Sobel J, Vajgel P (2010) Finding a needle in haystack: facebook's photo storage. OSDI 10:1–8
97. Fetterly D, Haridasan M, Isard M, Sundararaman S (2011) Tidyfs: a simple and small distributed file system. In: USENIX annual technical conference, pp 34–34
98. Quantcast file system. https://www.quantcast.com/wp-content/uploads/2012/09/QC-QFS-One-Pager2.pdf
99. Mapr file system. https://maprdocs.mapr.com/52/MapROverview/c_maprfs.html
100. Brewer E (2010) A certain freedom: thoughts on the cap theorem. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing, pp 335–335
101. Lourenço JR, Cabral B, Carreiro P, Vieira M, Bernardino J (2015) Choosing the right nosql database for the job: a quality attribute evaluation. J Big Data 2(1):1–26
102. Buyya R, Calheiros RN, Dastjerdi AV (2016) Big data: principles and paradigms. Morgan Kaufmann, ISBN-13: 9780128053942
103. Abadi D, Boncz P, Harizopoulos S, Idreos S, Madden S et al (2013) The design and implementation of modern column-oriented database systems. Now 5(3):197–280
104. Matei G, Bank RC (2010) Column-oriented databases, an alternative for analytical environment. Database Syst J 1(2):3–16
105. Floratou A, Patel JM, Shekita EJ, Tata S (2011) Column-oriented storage techniques for mapreduce. Proc VLDB Endow 4(7):419–429
106. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2):1–26
107. Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. ACM SIGOPS Oper Syst Rev 44(2):35–40
108. Stonebraker M, Abadi DJ, Batkin A, Chen X, Cherniack M, Ferreira M, Lau E, Lin A, Madden S, O'Neil E et al. (2005) C-store: a column-oriented DBMS. In: Proceedings of the 31st international conference on very large data bases, pp 553–564

109. Boncz PA, Zukowski M, Nes N (2005) Monetdb/x100: hyper-pipelining query execution. CIDR 5:225–237
110. Idreos S, Groffen F, Nes N, Manegold S, Mullender S, Kersten M (2012) Monetdb: two decades of research in column-oriented database architectures. Bull IEEE Comput Soc Tech Comm Data Eng 35(1):40–45
111. Sciore E (2007) Simpledb: a simple java-based multiuser syst for teaching database internals. ACM SIGCSE Bull 39(1):561–565
112. Zukowski M, Boncz P (2012) Vectorwise: beyond column stores. IEEE Data Eng Bull 35(1):21–27
113. Edward SG, Sabharwal N (2015) Mongodb limitations. In: Practical MongoDB, pp 227–232
114. Ravendb project. https://ravendb.net/docs/article-page/3.0/csharp
115. Cross datacenter replication. http://docs.couchbase.com/admin/admin/XDCR/xdcr-intro.html
116. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: amazon's highly available key-value store. ACM SIGOPS Oper Syst Rev 41(6):205–220
117. Basho products-riak products. http://basho.com/products/
118. Sumbaly R, Kreps J, Gao L, Feinberg A, Soman C, Shah S (2012) Serving large-scale batch computed data with project voldemort. In: Proceedings of the 10th USENIX conference on file and storage technologies, pp 18–18
119. Gudivada VN, Rao D, Raghavan VV (2014) NoSQL systems for big data management. In: 2014 IEEE World congress on services, pp 190–197
120. Allegrograph. https://franz.com/agraph/allegrograph/
121. Hypergraphdb. http://www.hypergraphdb.org/
122. Infinitegraph. http://www.objectivity.com/products/infinitegraph/
123. Moniruzzaman ABM, Hossain SA (2013) Nosql database: new era of databases for big data analytics-classification, characteristics and comparison. arXiv preprint arXiv:1307.0191
124. Apache hbase reference guide. https://hbase.apache.org/apache_hbase_reference_guide.pdf
125. Transparent data encryption. http://docs.datastax.com/en/archived/datastax_enterprise/4.0/datastax_enterprise/sec/secTDE.html
126. Khetrapal A, Ganesh V (2006) Hbase and hypertable for large scale distributed storage systems. Dept. of Computer Science, Purdue University, pp 22–28
127. Apache accumulo project. https://accumulo.apache.org/
128. Ghaffari Amir, Chechina Natalia, Trinder Phil, Meredith Jon (2013) Scalable persistent storage for Erlang: theory and practice. In: Proceedings of the twelfth ACM SIGPLAN workshop on Erlang, pp 73–74
129. Vogels W (2009) Eventually consistent. Commun ACM 52(1):40–44
130. Apache hbase project. https://blogs.apache.org/hbase/entry/hbase_cell_security
131. Mongodb mannual. https://docs.mongodb.org/manual/core/security-encryption-at-rest
132. Redis project. https://redis.io/
133. Random notes on improving the Redis LRU algorithm. http://antirez.com/news/109
134. Redis4.0. https://redislabs.com/blog/redis-4-0-0-released/
135. Redis cluster specification. https://redis.io/topics/cluster-spec
136. In-memory storage engine. http://learnmongodbthehardway.com/schema/wiredtiger/
137. The apache mahout project. https://mahout.apache.org/
138. Spark 2.3-mllib guide. https://spark.apache.org/releases/spark-release-2-3-0.html#mllib
139. Flinkml: Machine learning for flink. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/ml/
140. Mllib guide. https://spark.apache.org/docs/1.6.2/mllib-guide.html
141. Meng X, Bradley J, Yuvaz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S et al (2016) Mllib: Machine learning in apache spark. JMLR 17(34):1–7
142. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65
143. Machine learning library (mllib) guide. https://spark.apache.org/docs/latest/ml-guide.html
144. Different default regparam values in als. https://issues.apache.org/jira/browse/SPARK-19787
145. Spark 2.3, mllib guide. https://spark.apache.org/docs/2.3.0/ml-guide.html
146. Carbone P, Ewen S, Haridi S, Katsifodimos A, Markl V, Tzoumas K (2015) Apache flink: stream and batch processing in a single engine. Data Eng 38:28–38
147. Introducing Neo4j Bloom: Graph Data Visualization for Everyone. https://neo4j.com/blog/introducing-neo4j-bloom-graph-data-visualization-for-everyone/
148. Orange documentation https://orange.biolab.si/docs/

149. Raghavan UN, Réka A, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106
150. Chappell D (2015) Introducing azure machine learning. A guide for technical professionals, sponsored by microsoft corporation
151. Overview diagram of azure machine learning studio capabilities. https://docs.microsoft.com/en-in/azure/machine-learning/studio/studio-overview-diagram
152. Azure capabilities, limitations and support. https://docs.microsoft.com/en-us/azure/machine-learning/studio/faq
153. Ibm cloud/machine learning. https://console.bluemix.net/docs/services/PredictiveModeling/index.html#WMLgettingstarted
154. Amazon machine learning. https://aws.amazon.com/aml/
155. Amazon sagemaker features. https://aws.amazon.com/sagemaker/features/
156. Netflix's recommendation ml pipeline using apache spark. https://www.dbtsai.com/assets/pdf/2017-netflixs-recommendation-ml-pipeline-using-apache-spark.pdf
157. Role of spark in transforming ebay's enterprise data platform. https://databricks.com/session/role-of-spark-in-transforming-ebays-enterprise-data-platform
158. Number of full-time employees at alibaba from 2012 to 2017. https://www.statista.com/statistics/226794/number-of-employees-at-alibabacom/
159. Number of active consumers across alibaba's online shopping. https://www.statista.com/statistics/226927/alibaba-cumulative-active-online-buyers-taobao-tmall/
160. Huang L, Hu G, Lu X (2009) E-business ecosystem and its evolutionary path: the case of the alibaba group in china. Pacific Asia J Assoc Inf Syst 1(4)
161. A year of blink at alibaba: apache flink in large scale production. http://www.dataversity.net/year-blink-alibaba/
162. Gupta P, Sharma A, Jindal R (2016) Scalable machine-learning algorithms for big data analytics: a comprehensive review. Wiley Interdiscip Rev: Data Min Knowl Discov 6(6):194–214
163. Alibaba Blink: Real-time computing for big-time gains. https://medium.com/@alitech_2017/alibaba-blink-real-time-computing-for-big-time-gains-707fdd583c26
164. Ji X, Chun SA, Cappellari P, Geller J (2017) Linking and using social media data for enhancing public health analytics. J Inf Sci 43(2):221–245
165. Kanaujia PKM, Pandey M, Rautaray SS (2017) Real time financial analysis using big data technologies. In: I-SMAC (IoT in social, mobile, analytics and cloud)(I-SMAC), 2017 international conference on, pp 131–136
166. Moe WW, Schweidel DA (2017) Opportunities for innovation in social media analytics. J Prod Innov Manag 34(5):697–702
167. Psyllidis A, Bozzon A, Bocconi S, Bolivar CT (2015) A platform for urban analytics and semantic data integration in city planning. In: International conference on computer-aided architectural design futures, pp 21–36
168. Gust G, Flath C, Brandt T, Ströhle P, Neumann D (2016) Bringing analytics into practice: evidence from the power sector
169. Nguyen D, Lenharth A, Pingali K (2013) A lightweight infrastructure for graph analytics. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, pp 456–471
170. Baesens B, Van Vlasselaer V, Verbeke W (2015) Fraud analytics: a broader perspective. Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection, pp 313–346
171. Xu Z, Mei L, Chuanping H, Liu Y (2016) The big data analytics and applications of the surveillance system using video structured description technology. Cluster Comput 19(3):1283–1292
172. Bisias D, Flood M, Lo AW, Valavanis S (2012) A survey of systemic risk analytics. Annu Rev Financ Econ 4(1):255–296
173. Sagiroglu S, Sinanc D (2013) Big data: a review. In: Collaboration technologies and systems (CTS), 2013 international conference on, pp 42–47
174. Rabkin A, Arye M, Sen S, Pai VS, Freedman MJ (2014) Aggregation and degradation in JetStream: streaming analytics in the wide area. In: NSDI vol 14, 275–288
175. Zhang L, Stoffel A, Behrisch M, Mittelstadt S, Schreck T, Pompl R, Weber S, Last H, Keim D (2012) Visual analytics for the big data era comparative review of state-of-the-art commercial systems. In: Visual analytics science and technology (VAST), 2012 IEEE conference on, pp 173–182
176. Waller MA, Fawcett SE (2013) Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management. J Bus Logist 34(2):77–84
177. Chen H, Chiang RHL, Storey VC (2012) Business intelligence and analytics: from big data to big impact. MIS Q 36(4):1165–1188

178. Raghupathi W, Raghupathi V (2013) An overview of health analytics. J Health Med Inform 4(3):1–11
179. Cook DJ, Holder LB (2006) Mining graph data. Wiley, London
180. Fortunato S (2010) Community detection in graphs. Phys Rep 486(3):75–174
181. Xin RS, Gonzalez JE, Franklin MJ, Stoica I (2013) Graphx: a resilient distributed graph system on spark. In: First international workshop on graph data management experiences and systems 2(1–2):6
182. Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C (2011) Graphlab: A distributed framework for machine learning in the cloud. arXiv preprint arXiv:1107.0922
183. Introducing gelly: Graph processing with apache flink. https://flink.apache.org/news/2015/08/24/introducing-flink-gelly.html
184. Liu B (2007) Web data mining: exploring hyperlinks, contents, and usage data. Springer, Berlin. ISBN-13: 9783642194597
185. Wesley R, Eldridge M, Terlecki PT (2011) An analytic data engine for visualization in tableau. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data, pp 1185–1194
186. García M, Harmsen B (2012) Qlikview 11 for developers. Packt Publishing Ltd
187. JMP https://www.jmp.com/en_us/home.html
188. Microstrategy enterprise analytics and mobility. http://www.microstrategy.com/us/capabilities/visualizations
189. Tibco spotfire. http://spotfire.tibco.com/
190. Abousalh-Neto NA, Kazgan S (2012) Big data exploration through visual analytics. In: Visual analytics science and technology (VAST), 2012 IEEE conference on, pp 285–286
191. Sas. http://www.sas.com/en_in/home.html
192. Advizor. http://www.advizorsolutions.com/
193. Smoot ME, Ono K, Ruscheinski J, Wang P-L, Ideker T (2011) Cytoscape 2.8: new features for data integration and network visualization. Bioinformatics 27(3):431–432
194. Batagelj V, Mrvar A (1998) Pajek-program for large network analysis. Connections 21(2):47–57
195. Smith MA, Shneiderman B, Milic-Frayling N, Mendes Rodrigues E, Barash V, Dunne C, Capone T, Perer A, Gleave E (2009) Analyzing (social media) networks with NodeXL. In: Proceedings of the fourth international conference on communities and technologies, pp 255–264
196. Bastian M, Heymann S, Jacomy M et al (2009) Gephi: an open source software for exploring and manipulating networks. ICWSM 8:361–362
197. Csardi G, Nepusz T (2006) The igraph software package for complex network research. Int J Complex Syst 1695(5):1–9
198. Apache hadoop project. http://hadoop.apache.org
199. Sakr S, Liu A, Fayoumi AG (2013) The family of mapreduce and large-scale data processing systems. ACM Comput Surv 46(1):11
200. Lee K-H, Lee Y-J, Choi H, Chung YD, Moon B (2012) Parallel data processing with mapreduce: a survey. AcM sIGMoD Rec 40(4):11–20
201. Chen Y, Kreulen J, Campbell M, Abrams C (2011) Analytics ecosystem transformation: a force for business model innovation. In: 2011 Annual SRII global conference, pp 11–20
202. Venner J, Wadkar S, Siddalingaiah M (2014) Pro apache Hadoop. ISBN: 9781430248637
203. Apache hadoop project. http://hadoop.apache.org/docs/r2.5.2/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html
204. Hdfs high availability using the quorum journal manager. https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html
205. Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe Jason, Shah Hitesh, Seth Siddharth et al (2013) Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing, pp 5:1–16
206. HDFS Erasure Coding. http://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html
207. Apache Hadoop 3.0.1. http://hadoop.apache.org/docs/r3.0.1/
208. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. HotCloud 10:10–10
209. Marcu O-C, Costan A, Antoniu G, Pérez-Hernández MS (2016) Spark versus flink: understanding performance in big data analytics frameworks. In: Cluster computing (CLUSTER), 2016 IEEE international conference on, pp 433–442
210. Kubernetes concepts. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
211. Rensin DK (2015) Kubernetes-scheduling the future at cloud scale

212. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Antony S, Liu H, Murthy R (2010) Hive-a petabyte scale data warehouse using hadoop. In: 2010 IEEE 26th international conference on data engineering (ICDE 2010), pp 996–1005
213. Impala project. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-impala.html
214. Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, et al (2015) Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, pp 1383–1394
215. Traverso M (2013) Presto: interacting with petabytes of data at facebook. Retrieved February 4:2014
216. Hausenblas M, Nadeau J (2013) Apache drill: interactive ad-hoc analysis at scale. Big Data 1(2):100–104
217. Apache kylin. http://kylin.apache.org/docs
218. Ho L-Y, Li T-H, Wu J-J, Liu P (2013) Kylin: an efficient and scalable graph data processing system. In: Big data, 2013 IEEE international conference on, pp 193–198
219. Lamb A, Fuller M, Varadarajan R, Tran N, Vandiver B, Doshi L, Bear C (2012) The vertica analytic database: C-store 7 years later. Proc VLDB Endow 5(12):1790–1801
220. Chattopadhyay B, Lin L, Liu W, Mittal S, Aragonda P, Lychagina V, Kwon Y, Wong M (2011) Tenzing a SQL implementation on the mapreduce framework
221. Floratou A, Minhas UF, Özcan F (2014) Sql-on-hadoop: full circle back to shared-nothing database architectures. Proc VLDB Endow 7(12):1295–1306
222. Nasir MAU (2016) Fault tolerance for stream processing engines. arXiv preprint arXiv:1605.00928
223. Apache storm. http://storm.apache.org/
224. Apache storm. http://storm.apache.org/releases/current/Concepts.html
225. van der Veen JS, van der Waaij B, Lazovik E, Wijbrandi W, Meijer RJ (2015) Dynamically scaling apache storm for the analysis of streaming data. In: Big data computing service and applications (BigDataService), 2015 IEEE first international conference on, pp 154–161
226. Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J et al (2014) Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, pp 147–156
227. Apache strom 1.2.1. http://storm.apache.org/releases/current/Fault-tolerance.html
228. Storm 1.2.0. http://storm.apache.org/2018/02/15/storm120-released.html
229. Samza documentation. https://samza.apache.org/learn/documentation/0.14/comparisons/spark-streaming.html
230. Bockermann C (2014) A survey of the stream processing landscape. Lehrstuhl fork unstliche Intelligenz Technische Universit. at Dortmund
231. Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: Data mining workshops (ICDMW), 2010 IEEE international conference on, pp 170–177
232. Zaharia M, Das T, Li H, Shenker S, Stoica I (2012) Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. HotCloud 12:10–10
233. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, pp 423–438
234. Spark streaming programming guide. https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html#discretized-streams-dstreams
235. Improved fault-tolerance and zero data loss in apache spark streaming. https://databricks.com/blog/2015/01/15/improved-driver-fault-tolerance-and-zero-data-loss-in-spark-streaming.html
236. Apache spark 2.3. https://spark.apache.org/releases/spark-release-2-3-0.html
237. Chandy KM, Lamport L (1985) Distributed snapshots: determining global states of distributed systems. ACM Trans Comput Syst 3(1):63–75
238. Apache spark 2.3. https://databricks.com/blog/2018/02/28/introducing-apache-spark-2-3.html
239. Alexandrov A, Bergmann R, Ewen S, Freytag J-C, Hueske F, Heise A, Kao O, Leich M, Leser U, Markl V (2014) The stratosphere platform for big data analytics. VLDB J 23(6):939–964
240. Apache flink 1.4. https://ci.apache.org/projects/flink/flink-docs-release-1.4/concepts/runtime.html
241. Flink checkpointing. https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/stream/checkpointing.html
242. Exactly-once processing in samza. https://cwiki.apache.org/confluence/display/SAMZA/SEP-10+Exactly-once+Processing+in+Samza
243. De Morales GF, Bifet A (2015) Samoa: scalable advanced massive online analysis. J Mach Learn Res 16(1):149–153
244. Samoa project. https://samoa.incubator.apache.org/documentation/SAMOA-Topology.html
245. Apache samoa documentation. https://samoa.incubator.apache.org/documentation/Home.html

246. Akidau T, Balikov A, Bekiroğlu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P, Whittle S (2013) Millwheel: fault-tolerant stream processing at internet scale. Proc VLDB Endow 6(11):1033–1044
247. Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel JM, Ramasamy K, Taneja S (2015) Twitter heron: stream processing at scale. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, pp 239–250
248. Abadi D, Carney D, Cetintemel U, Cherniack M, Convey C, Erwin C, Galvez E, Hatoun M, Maskey A, Rasin A et al (2003) Aurora: a data stream management system. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, pp 666–666
249. Heron project. https://twitter.github.io/heron/docs/concepts/architecture/#metrics-manager
250. Structured streaming programming guide. https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
251. Flink streaming. https://ci.apache.org/projects/flink/flink-docs-master/dev/datastream_api.html
252. Fu M, Agrawal A, Floratou A, Graham B, Jorgensen A, Li M, Lu N, Ramasamy K, Rao S, Wang C (2017) Twitter heron: towards extensible streaming engines. In: Data engineering (ICDE), 2017 IEEE 33rd international conference on, pp 1165–1172
253. Amazon kinesis data streams. https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html
254. Azure stream analytics. https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction#how-does-stream-analytics-work
255. Ibm streaming analytics. https://www.ibm.com/cloud/streaming-analytics
256. Samza-storm. https://samza.apache.org/learn/documentation/0.7.0/comparisons/storm.html
257. Apache storm 2.0. http://storm.apache.org/releases/2.0.0-SNAPSHOT/index.html
258. Shukla A, Chaturvedi S, Simmhan Y (2017) Riotbench: a real-time iot benchmark for distributed stream processing platforms. arXiv preprint arXiv:1701.08530
259. Dreissig F, Pollner N (2017) A data center infrastructure monitoring platform based on storm and trident. Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband
260. Saha B, Shah H, Seth S, Vijayaraghavan G, Murthy A, Curino C (2015) Apache tez: a unifying framework for modeling and building data processing applications. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, pp 1357–1369
261. Tpc-h is a decision support benchmark. http://www.tpc.org/
262. Hortonworks data platform-apache hive performance tuning. https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.5/bk_hive-performance-tuning/bk_hive-performance-tuning.pdf
263. Aws-containers. https://aws.amazon.com/what-are-containers/
264. Apache mesos. http://mesos.apache.org/documentation/latest/
265. Sebastio S, Ghosh R, Mukherjee T (2018) An availability analysis approach for deployment configurations of containers. IEEE Trans Serv Comput
266. Medel V, Rana O, Bañares JÁ, Arronategui Unai (2016) Modelling performance and resource management in kubernetes. In: Utility and cloud computing (UCC), 2016 IEEE/ACM 9th international conference on, pp 257–262
267. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: NSDI, vol 11, pp 295–308
268. Amazon web services. https://aws.amazon.com/docker/
269. Kreps J, Narkhede N, Rao J et al (2011) Kafka: a distributed messaging system for log processing. In: Proceedings of the NetDB, pp 1–7
270. Rabbitmq. https://www.rabbitmq.com/
271. Activemq. http://activemq.apache.org/
272. AmazonmQ. https://aws.amazon.com/amazon-mq/
273. Lampesberger H (2016) Technologies for web and cloud service interaction: a survey. Serv Oriented Comput Appl 10(2):71–110
274. Dobbelaere P, Esmaili KS (2017) Kafka versus RabbitMQ. arXiv preprint arXiv:1709.00333
275. Sangat P, Indrawan-Santiago M, Taniar D (2018) Sensor data management in the cloud: data storage, data ingestion, and data retrieval. Concurr Comput: Pract Exp 30(1)
276. Hoffman S (2013) Apache flume: distributed log collection for hadoop. Packt Publishing Ltd
277. Ting K, Cecho JJ (2013) Apache Sqoop Cookbook. O'Reilly Media, Inc
278. Rabkin A, Katz RH (2010) Chukwa: a system for reliable large-scale log collection. LISA 10:1–15
279. Apach sqoop-overview. https://blogs.apache.org/sqoop/entry/apache_sqoop_overview
280. Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein JM (2010) Graphlab: a new framework for parallel machine learning. arxiv preprint. arXiv preprint arXiv:1006.4990
281. Aver C (2011) Giraph: large-scale graph processing infrastructure on hadoop. In: Proceedings of the Hadoop summit. Santa Clara 11(3), 5–9

282. Gonzalez JE, Low Y, Haijie G, Bickson D, Guestrin C (2012) Powergraph: distributed graph-parallel computation on natural graphs. OSDI 12(1):2–2
283. Salihoglu S, Widom J (2013) Gps: a graph processing system. In: Proceedings of the 25th international conference on scientific and statistical database management 22, pp 1–12
284. Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: graph processing in a distributed dataflow framework. OSDI 14:599–613
285. Xin RS, Crankshaw D, Dave A, Gonzalez JE, Franklin MJ, Stoica I (2014) Graphx: unifying data-parallel and graph-parallel analytics. arXiv preprint arXiv:1402.2394
286. Graphx programming guide. https://spark.apache.org/docs/latest/graphx-programming-guide.html
287. Junghanns M, Petermann A, Gómez K, Rahm E (2015) Gradoop: scalable graph data management and analytics with hadoop. arXiv preprint arXiv:1506.00548
288. Hunt P, Konar M, Junqueira FP, Reed B (2010) Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX annual technical conference 8(9)
289. Myriad home. https://cwiki.apache.org/confluence/display/MYRIAD/Myriad+Home
290. Apache avro. https://avro.apache.org/docs/current/
291. Hu W, Qu Y (2008) Falcon-AO: a practical ontology matching system. Web Semant: Sci Serv Agents World Wide Web 6(3):237–239
292. Apache nifi project. https://nifi.apache.org/
293. Islam M, Huang AK, Battisha M, Chiang M, Srinivasan S, Peters C, Neumann A, Abdelnur A (2012) Oozie: towards a scalable workflow management system for hadoop. In: Proceedings of the 1st ACM SIGMOD workshop on scalable workflow execution engines and technologies 4:1–4:10

T. Ramalingeswara Rao is currently working toward his Ph.D degree from the Department of Mathematics, Indian Institute of Technology Kharagpur, India. He received his M.Tech degree in Computer Science and Data Processing from Indian Institute of Technology Kharagpur, India, in 2010. His research interests include data mining and Big data analytics.

Pabitra Mitra is currently working as an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India. He received his Ph.D degree in Computer Science from Indian Statistical Institute, India, in 2003. He received his B.Tech degree from the Indian Institute of Technology Kharagpur, India in 1996. He was honored with Royal Society Indo-UK Science Network Award 2006, INAE Young Engineer Award 2008, IBM Faculty Award 2010. His current research areas include data mining, pattern recognition, machine learning, information retrieval. He has published more than 219 papers in International journals and conferences in these areas.

Ravindara Bhatt did his B.E. in Electronics and Communication from G.B. Pant Engineering College, Autonomous State Govt. Engineering College, Uttrakhand, India in 1996. He completed M.Tech in Information Technology from Guru Gobind Singh Indraprastha University, Delhi, India in 2005. Ravindara completed his Ph.D. from Indian Institute of Technology Kharagpur, India, in 2015. Presently he is an Assistant Professor in the Department of Computer Science Engineering and Information Technology at Jaypee University of Information Technology, Waknaghat, Solan, India. His research interests include Big data, communication networks, privacy and security, and developing machine learning algorithms for real-world applications.

A. Goswami received his M.Sc and Ph.D degree from Jadavapur University, India, in 1985 and 1992 respectively. In 1992, he joined the Indian Institute of Technology Kharagpur, India, where at present he is a Professor in the Department of Mathematics. He has published articles in JORS, EJOR, Computers and Mathematics with applications, Productional Planning and Control, Opsearch, International journal of Systems Science, The journal of Fuzzy Mathematics, Journal of Information and knowledge Management, International journal of uncertainty, Fuzziness and knowledge-based systems, Journal of Applied Mathematics and Computing, International Journal of Data Analysis Techniques and Strategies, knowledge and Information Systems, Expert systems and applications, International journal of Production Economics, etc. His research interests include data mining, Big data, cryptography, distributed and object-oriented databases, inventory management under fuzzy environment, optimization, database systems, data mining techniques under fuzzy environment, and information security. He has published more than 115 papers in international journals and conferences in these areas.

Affiliations

T. Ramalingeswara Rao[1] · Pabitra Mitra[2] · Ravindara Bhatt[3] · A. Goswami[1]

Pabitra Mitra
pabitra@cse.iitkgp.ernet.in

Ravindara Bhatt
ravibhatt749@gmail.com

A. Goswami
goswami@maths.iitkgp.ac.in

1 Theoretical Computer Science Group, Department of Mathematics, Indian Institute of Technology Kharagpur, Kharagpur 721302, India

2 Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, India

3 Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat 173234, India