Chapter 4
Big Data Storage

Abstract In this chapter, we focus on the storage of big data. We will review important issues including massive storage systems, distributed storage systems, and big data storage mechanisms. On one hand, the storage infrastructure need to provide information storage service with reliable storage space; on the other hand, it must provide a powerful access interface for query and analysis of large amount of data. Such a storage infrastructure generally consists of hardware infrastructure and storage mechanisms.

4.1 Storage System for Massive Data

Data storage refers to the storage and management of large-scale datasets, while achieving reliability and availability. A data storage system consists of two parts: infrastructure and data storage methods or mechanisms. The hardware infrastructure includes massive shared Information Communication Technology (ICT) resources utilized to feedback instant demands of tasks, and such ICT resources are organized in an elastic manner. The hardware infrastructure shall feature elasticity and dynamic reconfiguration, to be adaptive to different application environments. Data storage methods are deployed on the top of hardware infrastructure to maintain large-scale datasets. Storage systems shall be equipped with many interfaces, rapid query, or other programming models for analysis of or interaction with stored data.

The big data paradigm brings about an explosive growth of data. The sharp growth of data has stringent requirements on storage and management. Traditionally, data storage equipment is only auxiliary equipment of servers, and most of them store, manage, look up, and analyze data with structured RDBMSs. The GB, TB, to PB sharp growth of big data makes traditional storage equipment and management modes inadequate. Data storage equipment is becoming increasingly more important, and storage cost becomes the main expense of many Internet companies. Therefore, there is a compelling need for research on data storage.

M. Chen et al., *Big Data: Related Technologies, Challenges and Future Prospects*, SpringerBriefs in Computer Science, DOI 10.1007/978-3-319-06245-7_4,

A large number of storage systems emerge to meet the demands of big data. Existing storage technologies can be classified as DAS (Direct Attached Storage) and network storage, while network storage can be further classified into NAS (Network Attached Storage) and SAN (Storage Area Network).

In DAS, disc drives are directly connected with servers. Storage is a peripheral equipment, while, data management servers and all kinds of application software are matched with storage sub-systems (this way, I/O may stress system bandwidths). DAS applies to a few server environments but, when the storage capacity is increased, the efficiency of storage supply will be quite low and the upgradeability and expandability will be greatly limited. In case of server abnormality, data could not be acquired and stored resources and data could not be shared. DAS is mainly used in personal computers and small-sized servers, which only support such applications requiring low storage capacities and does not directly support multi-computer shared storage. Tap drivers and RAID (redundant array of independent disks) are classic DAS equipments.

Network storage is to utilize the original network or a specially designed storage network to provide users with a uniform information access and sharing services of information systems. Network storage equipment includes special data exchange equipments, disk array, tap library, and other storage media, as well as special storage software. It is characterized with mass data storage, limited data sharing, full utilization of data mining and information, data reliability, data backup and safety, as well as simplified and unified data management. In addition, network storage features very strong expandability, so as to provide information transmission rates suited for large data volume.

NAS is actually an auxiliary storage equipment of a network. It is directly connected to a network through a hub or switch, communicating with the TCP/IP protocol. NAS is geared to message passing, and transmits data in the form of files. NAS has two prominent features. First, on physical connection, it directly connects the storage equipment to a network and then hangs the storage at the rear end of a server, thus avoiding the I/O burden at the server. Second, technically, it reduces the movements of the disk actuator arm and thus reduces R/W delay. However, the structure of NAS shows that it is still a traditional server storage equipment in essence.

SAN focuses on data storage with a flexible network topology and high-speed optical fiber connections. It allows multipath data switching among any internal nodes. Data storage management is located in a relatively independent storage local area network, so as to achieve a maximum degree of data sharing and data management, as well as seamless extension of the system. From the organization of a data storage system, DAS, NAS, and SAN can all be divided into three parts: (a) disk array: it is the foundation of a storage system and the fundamental guarantee for data storage; (b) connection and network sub-systems, which provide connection among one or more disc arrays and servers; (c) storage management software, which handles data sharing, disaster recovery, and other storage management tasks of multiple servers.

4.2 Distributed Storage System

The first challenge brought about by big data is how to develop a large scale distributed storage system for strategic preservation of data and efficient data processing and analysis. To use a distributed system to store massive data, the following factors should be taken into consideration:

- *Consistency*: a distributed storage system requires multiple servers to cooperatively store data. As there are more servers, the probability of server failures will be larger. Usually data is divided into multiple pieces to be stored at different servers to ensure availability in case of server failure. However, server failures and parallel storage may cause inconsistency among different copies of the same data. Consistency refers to assuring that multiple copies of the same data are identical.
- *Availability*: a distributed storage system operates in multiple sets of servers. As more servers are used, server failures are inevitable. It would be desirable if the entire system is not serious affected with respect to serving the reading and writing requests from customer terminals. This property is called availability.
- *Partition Tolerance*: multiple servers in a distributed storage system are connected by a network. The network could have link/node failures or temporary congestion. The distributed system should have a certain level of tolerance to problems caused by network failures. It would be desirable that the distributed storage still works well when the network is partitioned.

Eric Brewer proposed a CAP [1, 2] theory in 2000, which indicated that a distributed system could not simultaneously meet the requirements on consistency, availability, and partition tolerance; at most two of the three requirements can be satisfied simultaneously. Seth Gilbert and Nancy Lynch from MIT proved the correctness of CAP theory in 2002. Since consistency, availability, and partition tolerance could not be achieved simultaneously, we can have a CA system by ignoring partition tolerance, a CP system by ignoring availability, and an AP system that ignores consistency, according to different design goals. The three systems are discussed in the following.

CA systems do not have partition tolerance, i.e, they could not handle network failures. Therefore, CA systems are generally deemed as storage systems with a single server, such as the traditional small-scale relational databases. Such systems feature single copy of data, such that consistency is easily ensured. Availability is guaranteed by the excellent design of relational databases. However, since CA systems could not handle network failures, they could not be expanded to use many servers. This is way most large-scale storage systems are CP systems and AP systems.

Compared with CA systems, CP systems ensure partition tolerance. Therefore, CP systems can be expanded to become distributed systems. CP systems generally maintain several copies of the same data in order to ensure a level of fault tolerance. CP systems also ensure data consistency, i.e., multiple copies of the same data

are guaranteed to be completely identical. However, CP could not ensure sound availability because of the high cost for consistency assurance. Therefore, CP systems are useful for the scenarios with moderate load but stringent requirements on data accuracy (e.g., trading data). BigTable and Hbase are two popular CP systems. BigTable is well-known since it was successful for managing the background data of Google's search engine. Because a lot of data in Google is structured data, BigTable mainly stores data with tables. Nevertheless, when a lot of information is put in a table, the table size will grow. Such information should be partitioned and stored separately. The table is usually highly sparse. Therefore, BigTable divides the columns into different Column Families, where every column family stores the same type of information. This way, similar data is stored together and the same type of information is processed in the same manner, making it easy for system users. In the same column family, new columns can be arbitrarily inserted, thus reducing the usage limit of BigTable to a great extent.

BigTable is designed in the way similar to GFS, a distributed file system of Google, where one Master and several Tablet Servers constitute a star structure in a system. The star structure has a single point of failure. The load of the Master server should be reduced in order to minimize Master errors. In BigTable, data transmission and data addressing do not involve the Master. Therefore the load of the Master is not high. In order to solve the problem of a single point of failure, BigTable adopts a Master election mechanism. In particular, it incorporates an asynchronous and consistent locking mechanism to ensure that exact one Master is elected every time based on the Paxos protocol [3].

Data in BigTable is sequenced in the lexicographic order of rows. During data modification, we shall insert a record in a sequential table, find a position to be inserted, and then move the original data to make room for the newly inserted data. This operation is very time-consuming. BigTable utilizes batch processing to solve this problem. Specifically, BigTable uses two tables to store data: it stores historical data with a big table and stores recently modified data with a very small table.when the recent data accumulates to a certain amount or after a certain amount of time, BigTable merges the recent data into the historical data. This approach greatly reduces the times that big tables are modified, since only small tables are frequently modified. The cost of data modification is thus reduced to a great extent. Therefore, this method mitigates the problem of high cost for data changes and increases the look-up speed for recently modified data.

AP systems,also ensure partition tolerance. However, AP systems are different from CP systems in that AP systems also ensure availability. However, AP systems only ensure eventual consistency rather than strong consistency in the previous two systems. Therefore, AP systems only apply to the scenarios with frequent requests but not very high requirements on accuracy. For example, in online SNS (Social Networking Services) systems, there are many concurrent visits to the data but certain amount of data errors are tolerable. Furthermore, because AP systems ensure eventual consistency, accurate data can still be obtained after a certain amount of delay. Therefore, AP systems may also be used under the circumstances with no stringent real-time requirements.

Dynamo and Cassandra are two popular AP systems. Cassandra, with sound expandability, is used for storing massive textual data by mainstream commercial online SNS companies, such as Facebook and Twitter. Specifically, Cassandra utilizes the Consistent Hash algorithm to randomly and evenly map Key spaces of user identifier spaces of servers to the same value domain space, and enables servers to manage user data corresponding to the Keys of adjacent mapped values. This way, dynamic changes at any servers in the systems only affect the data corresponding to a small segment of value domains undertaken by themselves. Mainstream SNSs utilize such distributed Key-Value storage approach, so as to better meet the demands of expandability of large-scale online SNS systems and load balance for the servers, and be adaptive to dynamic changes of systems.

In order to support the storage of textual data of users, Cassandra inherits the column family model of BigTable to aggregate data with similar features into a column family. What is different from BigTable is that Cassandra may expand the concept of column family to a super column family-the column family of column families. On Cassandra nodes, every column family corresponds to a MemTable, a resident memory. When nodes write data, it first writes data into MemTable. In proper occasions, e.g., memory space occupied by MemTable reaches the upper bound or after a fixed amount of time, MemTable is stored into a corresponding SSTable of a disk. SSTable has a large operation throughput because of its sequential writing approach. The system builds a local index for every block including every piece of data written in the disk and then Cassandra stores the index in the internal memory in the form of Bloom Filter compression [4]. Because the compressed index excludes the relative position of the block in the file system, Cassandra does not perform well with regard to random reading.

4.3 Storage Mechanism for Big Data

Considerable research on big data promotes the development of storage mechanisms for big data. Existing storage mechanisms of big data may be classified into three bottom-up levels: (a) file systems, (b) databases, and (c) programming models.

File systems are the foundation of the applications at upper levels. Google's GFS is an expandable distributed file system to support large-scale, distributed, data-intensive applications [5]. GFS uses cheap commodity servers to achieve fault-tolerance and provides customers with high-performance services. GFS supports large-scale file applications with more frequent reading than writing. However, GFS also has some limitations, such as a single point of failure and poor performances for small files. Such limitations have been overcome by Colossus [6], the successor of GFS.

In addition, other companies and researchers also have their solutions to meet the different demands for storage of big data. For example, HDFS and Kosmosfs are derivatives of open source codes of GFS. Microsoft developed Cosmos [7] to support its search and advertisement business. Facebook utilizes Haystack [8]

to store the large amount of small-sized photos. Taobao also developed TFS and FastDFS. In conclusion, distributed file systems have been relatively mature after years of development and business operation. Therefore, we will focus on the other two levels in the rest of this section.

4.3.1 Database Technology

The database technology has been evolving for more than 30 years. Various database systems are developed to handle datasets at different scales and support various applications. It is apparent that traditional relational databases cannot meet the challenges on categories and scales brought about by big data. NoSQL databases (i.e., non traditional relational databases) are becoming more popular for big data storage. NoSQL databases feature flexible modes, support for simple and easy copy, simple API, eventual consistency, and support of large volume data. NoSQL databases are becoming the core technology for of big data. We will examine the following three main NoSQL databases in this section: Key-value databases, column-oriented databases, and document-oriented databases, each based on certain data models.

4.3.1.1 Key-Value Databases

Key-value Databases are constituted by a simple data model and data is stored corresponding to key-values. Every key is unique and customers may input queried values according to the keys. Such databases feature a simple structure and the modern key-value databases are characterized with high expandability and smaller query response time higher than those of relational databases. Over the past few years, many key-value databases have appeared as motivated by Amazon's Dynamo system [9]. We will next introduce Dynamo and several other representative key-value databases.

Dynamo

Dynamo is a highly available and expandable distributed key-value data storage system. It is used to manage store status of some core services in the Amazon e-Commerce Platform. Amazon e-Commerce Platform provides multiple services and data storage that can be realized with key access. The public mode of relational databases may generate invalid data and limit data scale and availability. Dynamo can meet requirements of such applications with a simple key-object interface. The Dynamo interface is constituted by simple reading and writing of data items. Dynamo achieves elasticity and availability through the data partition, data copy, and object edition mechanisms.

The Dynamo partition plan relies on Consistent Hashing [10] to divide load for multiple main storage machines. With this mechanism, the mapping scope of a hash function is deemed as a fixed circular space or "ring" (i.e. the maximum hash value is followed by the minimum hash value). Every node in the system is assigned with a random value in the space and such random value represents its "position" in the ring. The position of every data item identified with a key, can be computed through the calculation of hash value of a keyword in the data item. Then, we get the first node clockwise, with the position larger than that of the data item. This way, every node shall be responsible for the region between the node and the previous node. The Consistent Hash has a main advantage that node passing only affects directly adjacent nodes and do not affect other nodes.

Dynamo copies data items to N sets of mainframe computers, in which N is a configurable parameter in order to achieve high availability and durability. It distributes every key word K into a coordinating node. The coordinating node is responsible for the copy of data items within its scope. Apart from storing all key words within its scope, the coordinating node shall copy N−1 successive nodes in the ring clockwise. This way, every node in the system will be responsible for a region between itself and the Nth former node.

Dynamo system also provides eventual consistency, so as to conduct asynchronous update on all copies. Before the update is applied to all copies, the *put()* call may return to the caller. Consequently, the data returned by the next *get()* call may not be the recently updated data. If there is no failure, the propagation delay of updating can be determined. However, in case of failure (e.g, server failures or network partition), the update may not propagate to all the copies until a large delay.

Voldemort

Voldemort is also a key-value storage system, which was initially developed for and is still used by LinkedIn. Key words and values in Voldemort are composite objects constituted by tables and images. The voldemort interface includes three simple operations: reading, writing, and deletion, all of which are confirmed by key words. Voldemort provides asynchronous updating concurrent control of multiple editions but does not ensure data consistency. However, Voldemort supports optimistic locking for consistent multi-record updating: in case of conflict between the updating and any other operations, the updating operation may exit. The vector clock used in Dynamo provides various data editions with sequencing. The data copy mechanism of Voldmort is the same as that of Dynamo. Voldemort may store data in RAM but allows data be inserted into a storage engine. It is worth noting that Voldemort supports two storage engines including Berkeley DB and Random Access Files.

The key-value database emerged a few years ago. Deeply influenced by Amazon DynamoDB, other key-value storage systems include Redis, Tokyo Canbinet and Tokyo Tyrant, Memcached and MemcacheDB, Riak and Scalaris, all of which provide expandability by distributing key words into nodes. Voldemort, Riak, Tokyo

Cabinet, and Memecached can utilize attached storage devices to store data in RAM or disks. Other storage systems store data at RAM and provide disc backup, or rely on copies and copy recovery to avoid the need for backup.

4.3.1.2 Column-Oriented Databases

The column-oriented databases store and process data according to columns other than rows. Columns and rows are segmented in multiple nodes to realize expandability. The column-oriented databases are mainly inspired by Google's BigTable. In this section, we first discuss BigTable and then introduce several derivative tools.

BigTable

BigTable is a distributed, structured data storage system, which is designed to process the large-scale (PB class) data among thousands commercial servers [3]. The basic data structure of BigTable is a multi-dimension sequenced mapping with sparse, distributed, and persistent storage. Indexes of mapping are key words of rows, key words of columns, and timestamps, and every value in mapping is an unanalyzed byte array. The key words of rows in BigTable are 64KB character strings, in which the rows are stored according to the lexicographical order and are continually segmented into Tablets, i.e. units of distribution and load balance. This way, read a short row of data can be highly effective, since it only involves communication with a small portion of machines. The columns are grouped according to the prefixes of key words, which are called column families. These column families are the basic units for access control. The timestamps are 64-bit integers to distinguish different editions of cell values. Clients may flexibly determine the quantity of cell editions to be stored. These editions are sequenced in the descending order of timestamps, so the latest edition will always be read.

The BigTable API features the creation and deletion of Tablets and column families as well as modification of metadata of clusters, tables, and column families, and access control rights. Client applications may write or delete values of BigTable, look up values from columns, or browse sub-datasets in a table. BigTable also supports some other characteristics, such as transaction processing in a single row. Users may utilize such features to conduct more complex processing on data.

BigTable is based on many fundamental components of Google, including GFS [5], cluster management system, SSTable file format, and Chubby [11]. GFS is use to store data and log files. The cluster management system is responsible for task scheduling, management of shared resources in machines, processing of machine failures, and monitoring of machine statuses. SSTable file format is used to store BigTable data internally. SStable provides mapping between persistent, sequenced, and unchangeable key words and values, with key words and values of any byte strings. BigTable utilizes Chubby for the following server tasks: (1) ensure there is at most one active Master copy at any time; (2) store the bootstrap location of

BigTable data; (3) look up Tablet server; (4) conduct error recovery in case of Table server failures; (5) store BigTable schema information; (6) store the access control table.

Every procedure executed by BigTable includes three main components: Master server, Tablet server, and client library. BigTable only allows one set of Master server be distributed to be responsible for distributing tablets for Tablet server, detecting added or removed Tablet servers, and conducting load balance. In addition, it can also modify the BigTable schema, e.g., creating tables and column families, and collecting garbage saved in GFS as well as deleted or disabled files, and using them in specific BigTable instances. Every tablet server manages a Tablet set and is responsible for the processing of loaded Tablet reading and writing, and segmenting Tablets when they are too big. The companying application client library is used to communicate with BigTable instances.

Cassandra

Cassandra is a distributed storage system to manage the huge amount of structured data distributed among multiple commercial servers [12]. The system was developed by Facebook and became an open source tool in 2008. It adopts the ideas and concepts of both Amazon Dynamo and Google BigTable, especially integrating the distributed system technology of Dynamo with the BigTable data model. Tables in Cassandra are in the form of distributed four-dimensional structured mapping, where the four dimensions including row, column family, column, and super column. A row is distinguished by a string-key with arbitrary length. No matter what the amount of columns to be read or written is, the operation on rows is an atomic operation. Columns may constitute clusters, which is called column families, which are similar to the data model of BigTable. Cassandra provides two kinds of column families: column families and super columns. The super column includes any quantity of columns with names related to the super column. A column family includes columns and super columns, which may be continuously added to the column family during execution. The partition and copy mechanisms of Cassandra are very similar to those of Dynamo, so as to achieve consistency.

Derivative Tools of BigTable

Since the BigTable code cannot be obtained through the open source license, some open source projects compete to implement the BigTable concept to develop similar systems, such as HBase and Hypertable.

HBase is a BigTable clone programmed with Java and is a part of Hadoop of Apache's MapReduce framework [13]. HBase replaces GFS with HDFS. It writes updated contents into the RAM and regularly writes them into files in discs. The row operations are atomic operations, equipped with row-level locking and transaction

processing. Large-scope transaction processing is provided with optional support. Partition and distribution are transparently operated, with client hash or a fixed secret key space.

HyperTable was developed similar to BigTable to obtain a set of high-performance, expandable, and distributed storage and processing systems for structured and unstructured data [14]. HyperTable relies on distributed file systems, e.g. HDFS, and distributed lock manager. Data representation, processing, and partition mechanism are similar to that in BigTable. HyperTable has its own query language, called HyperTable query language (HQL), and allows users to create, modify, and query underlying tables.

Since the column-oriented storage databases mainly emulate BigTable, their designs are all similar, except for the concurrency mechanism and several other features. For example, Cassandra emphasizes weak concurrency of concurrent control of multiple editions, while HBase and HyperTable focus on strong consistency through locks or log records.

4.3.1.3 Document Databases

Compared with key-value storage, document storage can support more complex data forms. Since documents do not follow strict modes, there is no need to conduct mode migration. In addition, key-value pairs can still be saved. We will examine three important representatives of document storage systems, i.e., MongoDB, SimpleDB, and CouchDB.

MongoDB

MongoDB is an open-source document-oriented database [15]. MongoDB stores documents as Binary JSON (BSON) objects [16], which is similar to object. Every document has an ID field as the main key word. Query in MongoDB is expressed with syntax similar to JSON. A database driver sends the query as a BSON object to MongoDB. The system allows query on all documents, including embedded objects and arrays. Indexes may be created for queryable fields in documents to enable rapid query.

The copy operation in MongoDB can be executed with log files in the main nodes that support all the high-level operations conducted in the database. During the copy operation, the machine queries all the writing operations since the last synchronization of the machine and executing operations in log files in local databases. MongoDB supports horizontal expansion with automatic sharing to distribute data among thousands of nodes by automatically balancing load and keep the system up and running in case of failure.

SimpleDB

SimpleDB is a distributed database and a web service of Amazon [17]. Data in SimpleDB is organized into various domains in which data may be stored, acquired, and queried. Domains include different properties and name/value pair sets of projects. Date is copied to different machines at different data centers in order to ensure data safety and improve performance. This system does not support automatic partition and thus could not be expanded with the change of data volume. SimpleDB allows users to use SQL to run query, e.g., selecting sentences nonconforming to a single domain. It is worth noting that SimpleDB can assure eventual consistency but does not feature MVCC (Muti-Version Concurrency Control). Therefore, conflicts therein could not be detected from the client side.

CouchDB

Apache CouchDB is a document-oriented database written in Erlang [18]. Data in CouchDB is organized into documents that consist of fields named by key words/names and values, and are stored and accessed as JSON objects. Every document is provided with a unique identifier. CouchDB allows access to database documents through the RESTful HTTP API. If a document needs to be modified, the client can download the entire document, modify it, and then send it back to the database. After a document is rewritten once, the identifier will be modified and updated. CouchDB utilizes the optimal copying to acquire scalability but without a sharing mechanism. Since various CouchDBs may be executed along with other transactions simultaneously, any kinds of Replication Topology can be built. The consistency type of CouchDB relies on the copy mechanism. If the server-server configuration is utilized, CouchDB system can ensure eventual consistency; with the master-slave configure, strong consistency can be assured. MVCC in CouchDB is synchronously executed with the historical Hash records.

Except for properties, sets, and indexes defined in sets, all documents are stored without a document schema. Generally, they do not provide explicit lock and, compared with traditional relational databases, feature weaker concurrency and atomic properties. Documents may be distributed to nodes of all systems to achieve scalability at different levels.

4.3.1.4 Platform for Nimble Universal Table Storage

Platform for Nimble Universal Table Storage (PNUTS) is a large-scale parallel geographically-distributed system for Yahoo!'s web applications [19]. It relies on a simple relational data model in which data is organized into a property record table. In addition to the classic data types, blob (binary large object or basic large object) is also an effective data type that allows any structures within records (not always large-scale binary objects such as images or audio frequency).

In the physical layer, the system is divided into different regions, each of which includes a set of complete system components and complete copies of tables. The data table is horizontally segmented into record groups, which are called Tablets. Tablets are distributed among many servers; every server may have tens of thousands of Tablets but a Tablet may only be stored in a region of a server. The query language of PNUTS supports the selection and projection of a signal table. To upgrade or delete an existing record, the main key words must be specified. The consistency mode provided by PNUTS has a feature between the general serializability and eventual consistency.

4.3.2 Design Factors

Of the various database systems, there is not a single system that can achieve the optimal performance under all workload circumstances. In each database system, some performance goals have to be compromised to achieve optimized operation for specific applications.

Cooper et al. in [20] discussed the trade-offs confronted by data management systems based on cloud computing, including reading performance and writing performance, delay and durability, synchronous and asynchronous copies, and data segmentation, among others. Some researchers also differentiated and analyzed other design factors [21–23]. In the following, we compare several prominent features of the existing database systems (rather than analyzing the design goals of a specific system).

- *Data Model*: this section examined three core data models, i.e. key-value, column, and document models. In particular, PNUTS uses a row-oriented data model.
- *Data Storage*: in some systems data are designed to be stored in RAM and their snapshots or copies are stored in discs. Other systems store data in discs, with the cache stored in RAM. A few systems have pluggable background programs that are allowed to use different data storage media, or standardized underlying document systems are required.
- *Concurrency Control*: there are three concurrency control mechanisms used in the existing systems: lock, MVCC, and non-concurrency control. The lock mechanism only allows a user to read or modify a real object (i.e., object, document, or row) at any time. The MVCC mechanism ensures the reading consistency. However, if several users modify a real object at the same time, several conflicting editions of a real object may be created. Some systems do not offer atomicity but allow different users to concurrently modify different parts of the same object, and may not ensure which edition will be acquired during reading.
- *Consistency*: according to the CAP theorem, strict consistency could not be simultaneously achieved along with availability and partition tolerance. The

weak consistency, eventual consistency, and time axis consistency of both types should be generally compromised for each other. Eventual consistency means that all the updating operations will finally be propagated through the system and all the copies will be eventually consistent beyond a given period of time. Time axis consistency means that all the copies of a given record will apply the updating operations following the same order.

- *CAP Option*: The CAP theorem indicates that a shared data system may achieve at most two properties, among consistency, availability, and partition tolerance. Databases based on cloud computing needs to copy data from different servers in order to handle system failure in some regions, which basically requires consistency and availability. This way, the trade-off between consistency and availability can be determined. At present, various weak consistency models [24] have been proposed to achieve reasonable system availability.

4.3.3 Database Programming Model

The massive datasets of big data are generally stored in hundreds and even thousands of commercial servers. Apparently, the traditional parallel models (e.g., Message Passing Interface (MPI) and Open Multi-Processing (OpenMP)) may not be adequate to support such large-scale parallel programs.

Some parallel programming modes have been proposed for specific fields. These models effectively improve the performance of NoSQL and reduce the performance gap between relational databases. Therefore, these models have become the cornerstone for the analysis of massive data.

4.3.3.1 MapReduce

MapReduce [25] is a simple but powerful programming model for large-scale computing using a large number of clusters of commercial PCs to achieve automatic parallel processing and distribution. In MapReduce, the computational workload are caused by inputting key-value pair sets and generating key-value pair sets. The computing model only has two functions, i.e., Map and Reduce, both of which are programmed by users. The Map function processes input and generates intermediate key-value pairs. Then, MapReduce will combine all the intermediate values related to the same key and transmit them to the Reduce function. Next, the Reduce function receives the intermediate key and its value set, merges them, and generates a smaller value set. MapReduce has the advantage that it avoids the complicated steps for developing parallel applications, e.g., data scheduling, fault-tolerance, and inter-node communications. The user only needs to program the two functions to develop a parallel application. The initial MapReduce framework did not support multiple datasets in a task. This shortcoming has been mitigated by some recent enhancements [26, 27].

Over the past decades, people have widely utilized the traditional relational databases to manage datasets. Consequently, programmers are familiar with the advanced declarative language of SQL, a relational database, for task description and dataset analysis. However, the succinct MapReduce framework only provides two nontransparent functions without the common operations (e.g., projects and filters). Therefore, programmers have to spend time on programming the basic functions, which are generally hard to maintain and reuse. Incorporating the SQL style in the MapReduce framework would be a promising solution. To this end, some advanced language systems have been proposed, e.g., the Sawzall [28] of Google, the Pig Latin [29] of Yahoo!, the Hive [30] of Facebook, and the Scope [3] of Microsoft, so as to improve the programming efficiency and user friendliness.

4.3.3.2 Dryad

Dryad [31] is a general-purpose distributed execution engine for processing parallel applications of coarse-grained data. The operational structure of Dryad is a directed acyclic graph, in which vertexes represent programs and edges represent data channels. Dryad executes operations on the vertexes in computer clusters and transmits data via data channels, including documents, TCP connections, and shared-memory FIFO. During operation, resources in a logic operation graph are automatically map to physical resources.

The operation structure of Dryad is coordinated by a central program called job manager, which can be executed in clusters or workstations of users. The user workstations can access clusters through the network. A job manager includes application codes and program library codes, in which application codes are used to build a job communication graph and the program library codes are used to arrange available resources. All kinds of data are directly transmitted between vertexes. Therefore, the job manager is only responsible for decision-making, which does not obstruct any data transmission.

In Dryad, application developers can flexibly choose any directed acyclic graph to describe the communication modes of the application and express data transmission mechanisms. In addition, Dryad allows vertexes to use any amount of input and output data, while MapReduce supports limited computing, with only one input set and generating only one output set. DryadLINQ [32] is the advanced language of Dryad and is used to integrate the aforementioned SQL-like language execution environment.

4.3.3.3 All-Pairs

All-Pairs [33] is a system specially designed for biometrics, bio-informatics, and data mining applications. It focuses on comparing element pairs in two datasets by a given function. The All-Pairs problem may be expressed as a three-tuples (Set A, Set B, and Function F), in which Function F is utilized to compare all elements in

Set A and Set B. The comparison result is an output matrix **M**. It is also called the Cartesian product or cross join of Set A and Set B.

All-Pairs is implemented in four phases: system modeling, input data distribution, batch job management, and result collection. In Phase I, an approximation model of system performance will be built to assist in deciding how much CPU is needed and how to conduct job partition. In Phase II, a spanning tree is built for data transmission, which is completed within logarithmic time. This way, the workload of every partition may effectively get input data. In Phase III, after the data flow is delivered to proper nodes, the All-Pairs engine will build a batch-processing submission for jobs in partitions, sequence them in the batch processing system, and formulate a node running command to acquire data in nodes. The user-defined items will be executed in proper partition jobs to generate results in batch, with the results displayed in the output matrix. In the last phase, as the batch processing system completes its jobs, the extraction engine will collect results and combine them in a proper structure, which is generally a single file list. In the list, all results are put in order. If the system hides the execution details from the user, the user may define the data and calculation requirements using the given interfaces.

4.3.3.4 Pregel

The Pregel [34] system of Google facilitates the processing of large-sized graphs, e.g., analysis of network graphs and social networking services. A computational task is expressed by a directed graph constituted by vertexes and directed edges, in which every vertex is related to a modifiable and user-defined value. Directed edges are related to their source vertexes and every edge is constituted by a modifiable and user-defined value and an identifier of a target vertex. After the graph is built, the program conducts iterative calculations, which is called supersteps among which global synchronization points are set until algorithm completion and output completion. In every superstep, vertex computations are parallel and every vertex executes the same user-defined function to express a given algorithm logic. Every vertex may modify its status and the status of its output edges, receive a message sent from the previous superstep, send the message to other vertexes, and even modify the topological structure of the entire graph. Edges are not provided with corresponding computations. Functions of every vertex may be removed by suspension. When all vertexes are in an inactive status without any message to transmit, the entire program execution is completed. The Pregel program output is a set consisting of the values output from all the vertexes. Generally speaking, the Pregel program output and input are an isomorphic directed graph.

Inspired by the aforementioned programming models, other researches have also focused on programming modes for more complex computational tasks, e.g., iterative computations [35, 36], fault-tolerant memory computations [37], incremental computations [38], and flow control decision-making related to data [39].

References

1. Eric A Brewer. Towards robust distributed systems. In *PODC*, page 7, 2000.
2. Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.
3. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.
4. James Manyika, McKinsey Global Institute, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute, 2011.
5. Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
6. Marshall K McKusick and Sean Quinlan. Gfs: Evolution on fast-forward. *ACM Queue*, 7(7):10, 2009.
7. Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, 2008.
8. Doug Beaver, Sanjeev Kumar, Harry C Li, Jason Sobel, Peter Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, volume 10, pages 1–8, 2010.
9. Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, volume 7, pages 205–220, 2007.
10. David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.
11. Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.
12. Avinash Lakshman and Prashant Malik. Cassandra: structured storage system on a p2p network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 5–5. ACM, 2009.
13. Lars George. *HBase: the definitive guide*. O'Reilly Media, Inc., 2011.
14. Doug Judd. hypertable-0.9. 0.4-alpha.
15. Kristina Chodorow. *MongoDB: the definitive guide*. O'Reilly, 2013.
16. Douglas Crockford. The application/json media type for javascript object notation (json). 2006.
17. James Murty. *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly Media, Inc., 2009.
18. J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide*. O'Reilly, 2010.
19. Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.
20. Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.
21. Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency rationing in the cloud: Pay only when it matters. *Proceedings of the VLDB Endowment*, 2(1):253–264, 2009.
22. Kimberly Keeton, Charles B Morrey III, Craig AN Soules, and Alistair Veitch. Lazybase: Freshness vs. performance in information management. *ACM SIGOPS Operating Systems Review*, 44(1):15–19, 2010.

23. Daniela Florescu and Donald Kossmann. Rethinking cost and performance of database systems. *ACM Sigmod Record*, 38(1):43–48, 2009.
24. Maarten Van Steen. Distributed systems principles and paradigms. *Network*, 4:20, 2004.
25. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
26. Spyros Blanas, Jignesh M Patel, Vuk Ercegovac, Jun Rao, Eugene J Shekita, and Yuanyuan Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 975–986. ACM, 2010.
27. Hung-Chih Yang and D Stott Parker. Traverse: Simplified indexing on large map-reduce-merge clusters. In *Database Systems for Advanced Applications*, pages 308–322. Springer, 2009.
28. Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.
29. Alan F Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of map-reduce: the pig experience. *Proceedings of the VLDB Endowment*, 2(2):1414–1425, 2009.
30. Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.
31. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, 2007.
32. Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, volume 8, pages 1–14, 2008.
33. Christopher Moretti, Jared Bulosan, Douglas Thain, and Patrick J Flynn. All-pairs: An abstraction for data-intensive cloud computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–11. IEEE, 2008.
34. Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.
35. Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst. Haloop: Efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1–2):285–296, 2010.
36. Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.
37. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.
38. Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A Acar, and Rafael Pasquin. Incoop: Mapreduce for incremental computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 7. ACM, 2011.
39. Derek G Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. Ciel: a universal execution engine for distributed data-flow computing. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, page 9, 2011.