Review of the Basic Technology of Big Data on Analysis and Storage Applied in Power Grids

Lin Qian¹,a, Jun Yu¹,b, Guangxin Zhu¹,c, Ruocheng Xie²,d, Zhengbo Chen²,e, Lin Wang¹,f, Hengmao Pang¹,g, Zhu Mei¹,h, Mingjie Xu¹,i, Haiyang Chen¹,j

¹State Grid Electric Power Research Institute, Nanjing, China
²State Grid Zhejiang Electric Power Company Information & Telecommunication Branch, Hangzhou, China

Abstract. With the era of big data coming, the amount of data generated by humans has grown exponentially, and people have become aware of the importance of data. This paper introduces the concept of big data and big data technology firstly, and then describes the mature technology of big data analysis: Hadoop and Spark. And Taking HBase as an example, the basic theory of big data storage technology is analyzed. The paper is designed to understand the concept of big data and current developments, explain the basic technologies of big data storage and analysis to provide a reference for more accurate cognition.

1. Introduction

Big data is not a new technology, a new product, but a new phenomenon, which is an inevitable feature of the development of the Internet. Currently, there is no uniform definition of the concept of big data. McKinsey defines big data as a collection of data that cannot be processed in a certain amount of time [1]. Wikipedia defines big data as: huge amount of data, massive data, which means that the amount of data involved is so large that it cannot be intercepted, managed, processed, and organized by humans in an acceptable time to transform to human-readable information [2, 3]. At present, the more unified view is the four characteristics of big data, namely 4V [4]: huge data size, wide variety of data (Variety), low data value density and fast data processing speed (Velocity).

As we concerned, the big data technology is not the technology that deals with big data problems, but the component technology that is applied in big data systems. Generally speaking, big data technology can be applied to the Hadoop distributed file system [5]. Distributed problems have always been the core in big data technology. For example, Hadoop and HBase both use the secondary index method to solve the logical addressing problem in distributed systems [6, 7]. Official of Spark recommended that it is a good way to use resource management framework Mesos container virtualization technology [8] to solve resource allocation problems. Zookeeper provides distributed
locks and distributed consistency API interfaces [9], which specifically deals with distributed consistency issues and so on.

Big data puts forward a higher requirement for the real-time and effectiveness of data processing. It is necessary to transform traditional data processing technologies to the technologies suitable for big data storage and analysis according to the characteristics of big data. This paper summarizes and analyzes some basic technologies and current research directions in big data systems.

2. Big data analysis technology
Hadoop is a completely open-source ecosystem for process of different data sources, including visualization, analysis, sharing, searching, sorting and organization [10]. It is the most popular open-source processing platform in the field of big data, which integrating distributed computing and storage are managed in [11]. Spark [12] is a big data processing framework built on speed, ease of use, and complex analytics, developed and released by the University of California at Berkeley in 2010.

2.1. Hadoop and its technical support
Hadoop implements a distributed file system (Hadoop Distributed File System, HDFS). HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware for applications with large data sets [13]. The core design of the Hadoop framework is HDFS and MapReduce [14]. HDFS provides storage for massive amounts of data, and MapReduce provides calculations for massive amounts of data. As a distributed data processing architecture, Hadoop consists of many elements, including HDFS, MapReduce, HBase, Hive, Zookeeper, etc. The core part is HDFS distributed data storage and MapReduce data coexistence processing mechanism. Here are a few important elements.

**HDFS** is the core sub-project of the Hadoop. It is the basis of data storage management in distributed computing. It is developed based on the need to access and process with large files or in streaming data mode, which can run on cost-effective commercial servers.

**MapReduce** is a parallel computing programming model [15] for parallel computing of large data sets (greater than 1TB). The core idea is: (1) dividing the problem firstly; (2) pushing the calculation to the datasets instead of pushing the data to the calculation, effectively avoiding the large amount of communication overhead generated during the transmission process [16]. It should be noted that MapReduce can run on the relational database management system DBMS. [17] Without HDFS. HDFS and several peripheral tools of the MapReduce computing model form the Hadoop 1.0 ecosystem together.

**Hadoop 2.0** gets the separation of the MapReduce computing model used in Hadoop 1.0 from the resource management functions. Slave nodes run on each node. At the same time, they manage containers on specific nodes, monitor the execution of a node, report resource availability to the master node; the master node is responsible for arbitration between all applications in the sy Currently Hadoop 3.0 has been released, and its optimization measures include HDFS erasable coding, multiple NameNode support, MR Native Task optimization, YARN cgroup-based memory and disk IO isolation, YARN container resizing, and so on.

**Zookeeper** is a distributed consistency solution in big data systems. It abstracts the distributed consistency problems that may be encountered in other components and processes them. Other components only need to call or implement the Zookeeper interface. Therefore, many components depend on Zookeeper, Hadoop HA [19], HBase, Kafka and other components of the Zookeeper instance.

2.2. Spark
**Spark** is just a computing platform that does not provide distributed storage and management itself. It must rely on Hadoop Distributed File System (HDFS) or some other storage method. So, Spark and Hadoop are not a relationship of exclusion and substitution. Instead, Hadoop provides features that Spark does not have, such as distributed file systems. And Spark provides real-time memory processing for data sets required. Spark has the concept of memory cluster computing, which can
cache data sets in memory to reduce access latency [20] at memory cluster computing. Spark also introduced an abstraction called Elastic Distributed Data Sets (RDD). RDD is a collection of read-only objects that are distributed across a set of nodes. These collections are flexible, and if a portion of the data set is lost, they can be reconstructed. The process of reconstructing a partial data set relies on a fault tolerance mechanism.

**Spark optimization** has several directions:

1. **Optimization based on business logic.** In different business scenarios, some spark operators or operator combinations may have large algorithm optimization space. However, this optimization is only an application in the development process and cannot be applied to all scenarios.

2. **Cache and memory-sorting optimization.** Spark's use of memory is still in its infancy and inefficient phase, and the efficiency of memory usage depends entirely on the quality of the code written by the programmer. The literature [21, 22] implements the automatic caching of RDD by analyzing the memory model and implementation code of spark, which is, the spark engine can automatically judge and cache valuable RDD, similar to the JVM's GC mechanism, making as little cache obsolescence as possible. At the same time, it can reduce program memory consumption and improve component usability. According to the semantic analysis of the code and the calculation of the value density of the RDD, the optimal rearrangement of the operation sequence is performed: the original LRU algorithm and the multi-level cache model are replaced with a register storage model configured according to the memory weight information. Through the above two optimizations, the operational efficiency of the task under limited resources and the stability of task efficiency in different cluster environments have been improved. However, the shortcoming of this paper is that there is no optimization of memory sharing problems between different nodes and different tasks.

3. **Configuration parameter optimization.** There are nearly 180 configuration parameters in Spark. These configuration parameters provide users with the space to configure dependencies and optimize according to business needs. Industrial practice and numerous experiments have shown that the setting of Spark configuration parameters has a great impact on the running performance of the platform (memory usage, disk IO, network IO, CPU occupy) [23]. At present, there are few literatures on parameter optimization. Ravi et al. [24] proposed a Spark parameter configuration and optimization method in Apache Big Data 2015. In addition, the Tuning Spark document and Cloudera on the official website of Apache Spark are also targeted Spark tuning method.

4. **Scheduling optimization.** The scheduling optimization method of Hadoop MapReduce computing framework has great reference value for Spark scheduling optimization. Based on the characteristics of framework scheduling, it is proposed to further summarize and analyze the Spark scheduling optimization from the perspective of resource allocation and task scheduling. Literature [25] summarizes and compares the scheduling optimization methods of MapReduce computing architecture by nine foreign research institutions, and can be applied to spark optimization. These documents focus on the following perspectives:

   Reduce network I/O. By finding a reasonable data organization mechanism and task execution mechanism, localizing the data as much as possible, and merging the near-mining.

   Balance the processing and localization of data processing. In some cases, you should choose the appropriate resource group slot for task assignment, rather than do the data localization as much as possible.

   Data skew. Data skew often results in utilization of system resources. For example, most Tasks execute faster, while a few Tasks execute longer, or wait for a long time to indicate insufficient memory, causing the execution fails. Or the storage of data is uneven. Strictly according to the data localization processing method, some resources are overloaded, while other resources are idle. At this point, to keep the processing efficiency of the data, the data processing needs to be reasonably transferred, and the reasonable network IO and storage is necessary.

   Starvation problem. The starvation state occurs in a task because there is another job with a higher priority. This problem often occurs when unfair scheduling is employed.
(5) Shuffle process optimization. MapReduce always aggregate a type of data with common characteristics to one node for computation. These data are distributed across the various storage nodes and are processed by the computing units of the different nodes. Generally speaking, the process of re-shaping the data and convening it to different nodes is Shuffle. Shuffle may develop a variety of problems in the implementation process: First, The amount of data processed by the Shuffle process will be large, and it is necessary to spread terabytes or even petabytes of data to hundreds or even thousands and tens of thousands of machines; Second, in order to aggregate the data to the right node location, you need to ensure that the data is stored in the right partition. Since the data size is larger than the node's memory, a large amount of data will be read from the hard disk into the memory at this stage. Multiple hard disk read and write will occur; third, in order to improve efficiency in the process of data transmission, the data is compressed in most cases. But the time consumption caused by data decompression may offset the advantages brought by data compression. In some cases, the compression algorithm may be better, so it is difficult to adjust the compression ratio and decompression time. Fourth, data needs to be transmitted through the network during the Shuffle process. Serialization and deserialization also affect the performance of Spark jobs. Therefore, in view of the above problems, the current research direction of shuffle process optimization mainly includes three solutions: the solution when data skew occurs in the shuffle process, the optimization of shuffle compression algorithm, and the shuffle memory optimization [26].

3. Big data storage technology

Big data storage can rely on mainframe, or distributed MySQL implemented with MySQL proxy, but more often use NOSQL. NOSQL is "Not Only SQL" referring to a non-relational database, which is a supplement to the relational database DBMS. In the web2.0 environment, the NOSQL database has sprung up under the challenge of high concurrent read and write, high availability and high scalability of the database system and the need for massive data storage and access.

3.1. Basic theory of storage

**CAP** is the foundation for building a NOSQL database management system. The three letters represent strong consistency, availability, and partition fault tolerance. According to the CAP theory, any data sharing can only satisfy at most two of the three characteristics. The relational database chooses A and C, and the NOSQL database makes trade-offs between consistency and availability based on demand.

**BASE** is to improve the performance and usability of the system by sacrificing consistency and independence. This is the concept of the BASE method [27]. BASE includes: Basically Available, Soft-State, and Eventual Consistency. BASE theory is the evolution of CAP theory, which is completely different from the ACID characteristics of relational databases. It achieves basic consistency and flexible reliability by sacrificing strong consistency, and achieves final consistency to improve usability and system performance [28]. NOSQL generally follows the BASE principle, and these features are usually guaranteed by the distributed consistency component named zookeeper.

3.2. Storage model

There are many classifications of NoSQL databases, mainly including the following four typical types [29]: key value storage data model, columnar storage data model, and document storage data model.

**The key-value storage data model** uses an associative array (hash table) to provide a data model for high throughput data services. Key-value is a mapping. The system adds, queries, and deletes data through a unique key. Value is the stored content. The key-value storage database is fast to find the data unstructured and simple. However, it is not suitable for querying and updating bulk data, nor does it support data operations with particularly complex logic.

The columnar storage data model is a data model that stores data in columns. Columnar storage improves storage space utilization and query efficiency by storing the same column of data as much as possible on the same page of the hard disk, while supporting the "column family" (multiple columns
and one group, column family) feature, and saving a lot of I/O operations [30]. The column storage data model finds data fast and it is scalable, but it has low written efficiency and poor data integrity.

Document storage data model stores data as a document. The main goal of the document database is to bridge the gap between key-value storage and traditional relational data models. The data is usually stored in JSON or JSON-like (e.g. XML, BSON, etc.) documents [31]. Usually the query is more efficient.

### 3.3. HBase

HBase (Hadoop DataBase) is a high-reliability, high-performance, column-oriented, scalable distributed storage system. It provides a similar Big Table with the computing and storage ability of Hadoop [32].

The HBase architecture conforms to the server architecture and consists of two nodes, Master and RegionServer, as shown in Figure 3.1.

![Architecture of HBase](image)

**Fig. 1** Architecture of HBase

The main functions of the Master are: Coordinating and monitoring the RegionServer server to achieve load balancing; managing and allocating online regions, reassigning regions for load balancing and region recovery; creating, managing, and updating tables.

RegionServer runs on the HDFS DataNode and contains the following components: WAL, a file on a distributed file system that stores data on the persistence layer to recover data in the event of a system failure; BlockCache, store hot data in memory, and use LRU algorithm to eliminate data; MemStore, a write cache, to store new data that has not been written to disk, and to sort data before writing to disk. Under each region a column family corresponds to a MemStore; StoreFile, which is responsible for storing actual data, is the smallest storage unit in HBase.

The hotspots of HBase optimization research are as follows:

1. HBase secondary index query engine [33]

The secondary index implemented by HBase is the rowkey primary index, and the customized index data is the secondary index. With the secondary index, the query requirement of the non-rowkey component can be realized and the query performance can be improved.

The specific implementation method is that the client inputs the data to the Region Server (hereinafter referred to as RS) through the native HTable class. Then the Observer function of the Coprocessor of the RS terminal intercepts the put/delete instruction, and generates the index data configured in advance through the index constructor; the query is implemented by the End Point. The client sets the query condition through the custom API and submits it to the RS. The server firstly analyzes the condition information submitted by the client, and selects the optimal index through the query decision maker to query.
This method has a big drawback. You must use the Region pre-segment table and you need to set the prohibition of RegionSplit. This is because in order to improve the query speed, the index data and the query data are usually stored in a Region; When the Region is split, the split is usually uncontrollable, and there is no guarantee that the index data and the query data are still stored in the same Region.

(2) HBaseRegionServer loading balanced algorithm
The default rootRegion allocation policy can make the amount of data not too large in each server. But the frequency of data access is different. So, the default allocation strategy can easily cause server hotspot problems. In response to the hot issues of RegionServer, some studies have achieved good results through prediction and load balancing [34].

(3) HFile merging and segmentation strategy
HFile merge strategy, also known as Compaction, is an important part of HBase storage mechanism. Its core is to reduce disk IO and improve system reading performance by reducing the number of disk data files and deleting useless data. But in a merge, if too many merged files are selected, disk IO will be increased, which will affect system performance. In addition, with the operation of HBase, the more useless data in the file, the lower the efficiency of data query. There are two kinds of compaction in HBase: Minor and Major. Minor Compaction is the merging of some files under the column family and the collation of expired version data. Major Compaction is the merging of all files under the column family and the deletion and cleaning of multi-version data. Therefore, the strategy and implementation of Major Compaction greatly affect the stability of the system.

Data Tiered Compaction Policy is a merging strategy for hot data studied by Facebook developers. This strategy divides the latest hotspot data into HFile with smaller data volume and HFile with larger data volume for Friendship Circle data service. In this way, a large number of hot data will not exist in an HFile, thus avoiding the hot issue of HFile access.

(4) HBase read cache mechanism
Similar to other databases, optimizing IO read performance is a particular concern of HBase. The basic way to improve database IO is the caching mechanism. At present, there are three ways to implement the read cache of HBase: LRU BlockCache, SlabCache and BucketCache. LRU BlockCache is the default implementation of memory cache based on LRU; SlabCache uses out-of-heap memory to solve the STW problem, however causes low memory utilization. BucketCache expands SlabCache to allocate cache to high-speed disks such as SSD. It is worth mentioning that BucketCache was developed by Ali middleware and contributed to apache. At present, enterprises generally adopt LRUBlockCache + BucketCache two-level caching method. The main research direction lies in setting dynamic bucket size and setting dynamic bucket recovery mechanism to reduce memory usage reasonably [35].

4. Conclusions
Many big data technologies have moved from academic and theoretical to industrial and practical, and recently there has been significant progress in big data components. However, there are still many open issues such as ease of use, security, visualization, stability, scalability, data organization and analysis, resource management, privacy issues, legal issues, and so on. The distributed problem is a major research direction of current big data storage and analysis technology. This paper focuses on the basic technology of big data analysis and storage. Then it researches and analyzes the optimization method aiming at improving the execution efficiency of the spark engine, the HBase optimization strategy aiming at improving the HBase space storage and access efficiency. At last it points that the current important research direction of big data storage technology.

Acknowledgments
This work was financially supported by the science and technology projects of headquarters of State Grid Corporation (Research on Key Technologies of Autonomous and Controllable Distributed Hyper-converged IT Infrastructure Platform).
References

[1] RuiTong W, WeiChun L. Research on Technology of Basic Large Data Storage System [J]. Computer Technology and Development, 2016.
[2] XiaoFeng M, Xiang C. Big Data Management: Concepts, Techniques and Challenges [J]. Computer Technology and Development, 2013.
[3] JianGuang M, Wei J. The Concept, Characteristics and Application of Big Data [J]. Defense Technology Review, 2012.
[4] Peng L, ZhaoFeng W, Guyu H. Big Data is Undergoing Profound Changes [J]. ZTE Technology Journal, 2013.
[5] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system [C]. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.
[6] Zhou W, Li R, Yuan S, et al. MetaSpark: a spark-based distributed processing tool to recruit metagenomic reads to reference genomes [J]. Bioinformatics, 2017.
[7] Mundy J, Thorneithwaite W. The Microsoft data warehouse toolkit: with SQL Server 2008 R2 and the Microsoft Business Intelligence toolset [M]. John Wiley & Sons, 2011.
[8] Whitehorn M. Business Intelligence: The IBM Solution: Datawarehousing and OLAP [M]. Springer Science & Business Media, 1999.
[9] 9. https://yq.aliyun.com/articles/205946
[10] Heinrich J, Luo Y, Kirkpatrick A, et al. Evaluation of a Bundling Technique for Parallel Coordinates [J]. Energy Conversion & Management, 2011.
[11] XinHua D, RuiXuan L. Performance Optimization and Feature Enhancements of Hadoop System [J]. Journal of Computer Research and Development, 2013.
[12] Zaharia M, Chowndhury M, Franklin M J, et al. Spark: cluster computing with working sets [J]. Book of Extremes, 2010.
[13] Lin C. On the Application of Hadoop Platform in Big Data [J]. Journal of Taiyuan University (Natural Science Edition), 2017.
[14] Min Z, QiuYan T. The Status Quotation and Prospect of Big Data Application [J]. China Computer & Communication, 2019.
[15] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [J]. Communications of the ACM, 2008.
[16] Mcdonnell K T, Mueller K. Illustrative parallel coordinates [J]. Computer Graphics Form, 2008.
[17] XiongPai Q, HuiJu W, XiaoYong D. Big Data Analysis—Competition and Symbiosis of RDBMS and MapReduce [J]. Journal of Software, 2012.
[18] Kevin T. S, Boris L, Alexey Y. Hadoop Advanced Programming - Building and Implementing Big Data Solutions [M]. Tsinghua University Press, 2014.
[19] Apache Hadoop [EB/OL]. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html, 2018.
[20] LiBing W, Xin Q, LuYao Y, et al. Research on SQL-on-Hadoop systems [J]. Journal of Central China Normal University (Natural Sciences), 2016.
[21] Lin F. Research and Implementation of Memory Optimization Based on Parallel Computing Engine Spark [D]. Tsinghua University, 2018.
[22] Churila S A, Zhou G L, Shi L, et al. Parallel cube computing in Spark [J]. Journal of Computer Applications, 2016.
[23] Li M, Tan J, Wang Y, et al. Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark [C]. Proceedings of the 12th ACM International Conference on Computing Frontiers. ACM, 2015.
[24] Ravi N. Configuring and optimizing Spark applications with easeNishkam ravi, Cloudera [EB/OL]. https://apachebigdata2015.shed.org/event, 2015.
[25] Rumi G, Colella C, Ardagna D. Optimization Techniques within the Hadoop Ecosystem: A Survey [C]. 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE, 2014.
[26] Chern Y Z. Analysis and optimization of Memory Scheduling Algorithm of Spark Shuffle [D]. HangZhou: Zhe jiang University, 2016.

[27] Qing L. Processing of big data based on NoSql [D]. Xidian University, 2014.

[28] QingXue D. Analysis of NoSQL technology based on big data environment [J]. Electronic Design Engineering, 2016.

[29] JiaHeng L. Big Data Challenge and NoSQL Database Technology [M]. Beijing: Publishing House of Electronics Industry, 2014.

[30] Strauch C. NOSQL Databases [EB/OL]. http://www.christof-strauch.de/nosqldb.pdf.

[31] WoQI M. 7 key technologies of modern data architecture [J]. Computer Knowledge and Technology, 2018.

[32] Bigtable. Bigtable: a distributed storage system for structured data. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et al. Proceedings of OSDI 06: Seventh Symposium on Operating System Design and Implementation. 2006.

[33] Hong G, JianQian Z, YingYing Z, Kun G. HBase Secondary Index based on Coprocessor [EB/OL]. Computer Engineering and Applications. 2018.

[34] Fang S, Yong W. The Analysis and Optimization of Load Balancing Algorithm for Big Data Platform Based on HBase [EB/OL]. kns.cnki.net/KCMS/detail/42.1671.TP.20181226.1204.010.htm, 2018.

[35] YunPing W. The Research of HBase Compaction and BucketCache [EB/OL]. kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CMFD&dbname=CMFD201701&filename=1017700310, 2017.