Huawei Certification Big Data Course

HCIA-Big Data V3.5 (For Trainees)

ISSUE: V3.5

HUAWEI TECHNOLOGIES CO., LTD

Trademarks and Permissions

 and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://e.huawei.com

Huawei Certification System

The Huawei certification system is a platform for shared growth, part of a thriving partner ecosystem. There are two types of certification: one for ICT architectures and applications, and one for cloud services and platforms. There are three levels of certification available:

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

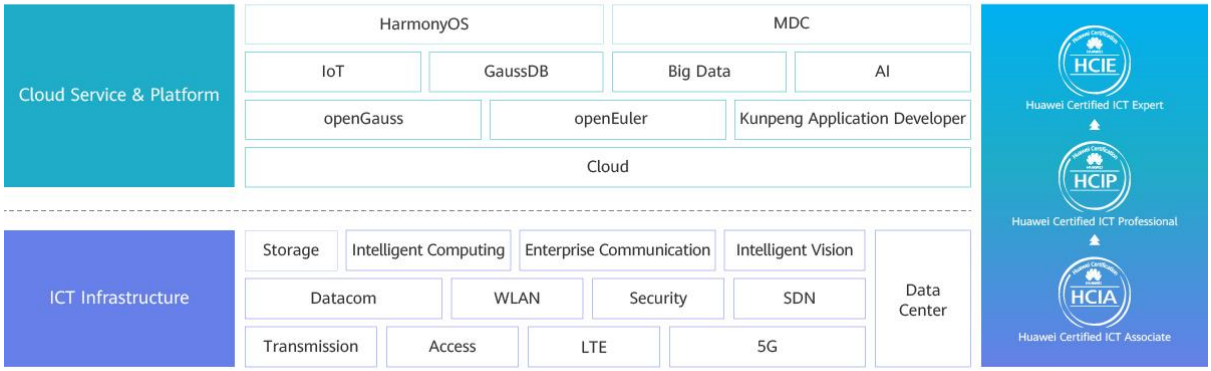
Huawei Certification covers all ICT fields and adapts to the industry trend of ICT convergence. With its leading talent development system and certification standards, it is committed to fostering new ICT talent in the digital era, and building a sound ICT talent ecosystem.

HCIA-Big Data V3.5 certification is intended to train and certify big data engineers who are capable of using Huawei's big data development platform MRS.

Passing the HCIA-Big Data Engineer V3.5 certification means that you already master the technical principles and architectures of big data components, including HDFS, Hive, HBase, ClickHouse, MapReduce, Spark, Flink, Flume and Kafka, capable of using Huawei big data platform MRS, and are able to operate and develop services based on Huawei MRS.

The Huawei certification system helps you embrace the up-to-date industry knowledge and trends, opens up a new horizon for you, and assists you during your pursuit for a new self.

Huawei Certification



Contents

1 Big Data Development Trends and the Kunpeng Big Data Solution

1.1 Big Data Era

1.1.1 Background

In the 1760s, the UK took the lead in launching the first industrial revolution, bringing the world into the age of steam. As a result, the UK became the first industrialized country in the world and brought about a series of social changes. In the 1860s, the second industrial revolution began. With the large-scale use of alternating current, humanity entered the age of electricity. The third scientific and technological revolution, marked by the invention and application of atomic energy, electronic computers, space technology, and bioengineering, involves many fields, such as information, new energy, new materials, biotechnology, space, and oceans. After completing the third industrial revolution, America has come to lead the world in technology.

The fourth technological revolution is materializing. It is centered on emerging IT technologies such as cloud, big data, IoT, and intelligence. Major economies around the world have adopted data openness as a national strategy and issued related data development strategies to promote future economic development. China has begun to deploy the big data industry in top-level design.

In addition to national strategic requirements, operators need to transform their mindsets to adapt to the digital and information era. We need to become not only a data manager, but also a data operator, because data drives user experience improvement, accurate decision-making, and process simplification.

1.1.2 What Is Big Data

Currently, there is no universally accepted definition of big data. Different definitions are all based on characteristics of big data. Wikipedia defines big data as data sets with sizes beyond the ability of commonly used software tools to capture, manage, and process data within a tolerable elapsed time. The research institute Gartner defines big data as high-volume, high-velocity, and high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation. Unlike traditional BI OLAP or data warehouse applications, big data analysis processes large data volumes and involves complex query and analysis. The four characteristics of big data defined in the industry are Volume, Variety, Velocity, Veracity.

- Volume

The most obvious feature of big data is its huge data volume. Currently, the data storage unit ranges from GB to TB, and even PB-scale. With the rapid development of networks and information technologies, data continues to grow explosively.

- Variety

  The second characteristic of big data is the diversity of data types. This is mainly reflected in diverse data sources and data formats. Currently, there are three types of data: 1. Structured data, such as that of financial systems, information management systems, and medical systems; 2. Unstructured data, such as video, images, and audio; 3. Semi-structured data, such as HyperText Markup Language (HTML), documents, emails, and web pages.

- Velocity

  The third characteristic is the speed of processing data analysis. In the big data era, data is time-sensitive. As time goes by, data value decreases. To mine data value as much as possible, algorithms help quickly process, analyze, and return data to users to meet their real-time requirements.

- Value

  In contrast with traditional small data, the biggest value of big data comes from mining valuable data for future trend and pattern prediction from a large amount of irrelevant data of various types. This is similar to gold miners extracting gold from massive amounts of sand.

1.1.3 Big Data Analysis vs. Traditional Data Analysis

Big data analysis differs greatly in multiple aspects from traditional data analysis. In terms of data scale, most traditional data analysis uses databases to store data, and the data processing scale is measured in MB. The processing scale of big data is measured in GB, TB, and PB. The data type changes from single structured data to unstructured, semi-structured, and structured data.

If we compare data to "fish", traditional data analysis is similar to "fishing in ponds" and big data analysis is similar to "fishing in the sea". Traditional data analysis and processing tools are simple but highly applicable. In addition, data is processed after models are available. However, there is no one-for-all tool for big data analysis. The tool may need to be replaced as the processed data changes. In addition, the data processing model also changes as data increases.

Currently, big data is booming. At the Sixth World Internet Conference on October 20, 2019, China released the *China Internet Development Report 2019*. According to the report, by the end of June 2019, the number of Chinese Internet users was 854 million and the number of websites was 5.18 million. China has become one of the largest generators of data in the world, with the most abundant data types. With the generation of massive data, industry applications based on big data are also changing, such as smart transportation, smart factory, smart government, smart finance, and smart security.

1.2 Big Data Application Fields

Today, data has penetrated into every industry and business domain, and social networking and IoT technologies have expanded data collection channels. Distributed storage and computing technologies are the technical basis of big data processing. Emerging technologies, such as neural networks, have ushered in a new era of big data analytics, laying a solid foundation for big data. Therefore, if enterprises want to leverage data in industry applications, they need to learn how to use it to gain business insight, predict trends, and guide efforts towards clear future targets.

The telecom, finance, and government industries mainly use structured and semi-structured data to complete operations, management, and supervision, and have high requirements for data analysis. For the Internet, which is mainly unstructured data, some enterprises have started to use new technologies to process big data with low value density.

According to the industry application scenarios from 2018 to 2019 provided by big data consulting companies, enterprise big data applications mainly focus on marketing analysis, customer analysis, and internal operation management. Currently, big data applications focus on how to improve profitability and operation efficiency for enterprises, while reducing operation costs. The following describes the application of big data in several important fields:

- Finance

  The application of big data in the financial field brings two changes. One is the change in customer experience. Before the emerging of big data, customers could only obtain financial services at a fixed time and place, and they received information passively. With big data, customers can obtain financial services through mobile devices anytime and anywhere. In addition, they can interact with products on mobile devices. The other is a change in enterprise services. Traditional financial institutions provide standardized services and contact customers through customer managers in a passive manner. Therefore, the interaction channels are limited. Financial institutions under big data can effectively streamline the communication between marketing, service, and operation through data analysis and mining, improving customer service experience and product value.

- Education

  Big data can be applied to education reform, learning analysis, and exam evaluation. In education reform, big data provides support for basic and higher education by analyzing students' psychological activities, learning behaviors, exam scores, and critical information about students' career planning. Many teaching data is now stored for statistical analysis by government agencies such as National Center for Education Statistics (NCES). The ultimate goal of big data analytics is to improve students' learning performance and provide personalized services for improving students' academic performance. It can improve students' academic performance and the rate of admission into higher schools, prevent them from skipping classes or even dropping out of school, and guide balanced education development.

  Nowadays, big data is not only used to display students' test scores and homework completion progress. Instead, teachers can view students' browsing of digital learning resources, submission of electronic homework, interaction between online

teachers and students, and completion of exams and tests. In these cases, big data allows the learning analysis system to continuously and accurately analyze the data of each student's participation in teaching activities. Teachers can quickly diagnose problems with more detailed information, such as the time and frequency of readings, give suggestions for improvement, and predict students' academic performance.

For examination evaluation, big data requires educators to update and transcend traditional ideas. That is, they need to also focus on how students behave during the teaching process. How long do students spend on each question in an exam? What's the maximum time? What's the minimum time? What's the average time? For questions that have been asked before, have the students answered the questions correctly? What clues in a problem that have benefited the students? By monitoring the information and providing students with personalized learning schemes and learning methods through the self-adaptive learning system, personal learning data archives can be generated to help educators understand the entire process of learning for students to master the course content and implement custom teaching in accordance with the students' aptitudes.

- Government and public security

  In the government and public security field, big data can be used to monitor population flows and generate prompt warnings, so that administrative departments can be notified of emergencies such as abnormal population flows. Cloud computing and massive data can also be used to locate areas that are most vulnerable to criminals and create a hotspot map of areas with high crime rates. When studying the crime rate in a certain district, various factors in the adjacent district are taken into consideration to provide support for the police departments to locate the high crime areas and catch suspects.

- Transportation planning

  Traffic management centers can dynamically monitor the traffic status of roads and hubs based on big data technologies to comprehensively obtain the information such as the road conditions and passenger traffic in key hubs such as railway stations. This provides data support for related departments for emergency response plans. Furthermore, traffic management departments can use big data to analyze and judge road safety situations and clear congested roads in a timely manner. Roads where accidents frequently occur and potential security risks exist are strictly managed and controlled. Meanwhile, warnings and related road traffic control information are released promptly during bad weather to ensure traffic safety.

- Clean energy

  In the nine-day period from June 20 to 28, 2018, Qinghai province used clean energy such as water, wind, and light to provide power supply for 216 hours, achieving zero emission of electricity and promoting the new practice of full-clean energy power supply with the goal of maximizing the consumption of new energy. Adjusting the peak load caused by the high proportion of new energy is the basis and key of "9-Day Green Power Supply" activity that uses the multi-energy complementary coordinated control technologies and a big data platform to improve new energy management. A compensation mechanism and load participation mechanism are introduced to the peak adjustment to expand the space for PV absorption.

Furthermore, multi-energy coordinated control technologies are under in-depth development and the construction of new energy big data centers are under construction to give priority to new energy sources and prolongs the utilization of power supply.

1.2.1 Big Data Computing Tasks

Big data can be stored, obtained, processed, and analyzed with many methods. Each big data source has different characteristics, including the data frequency, amount, processing speed, type, and authenticity. When big data is processed and stored, more dimensions are involved, such as governance, security, and policies.

Big data computing tasks are classified into I/O-intensive tasks, CPU-intensive tasks, and data-intensive tasks. Each type of computing task has its own characteristics.

I/O-intensive task: The CPU usage is low because most tasks are waiting for the completion of I/O operations. During the execution of I/O-intensive tasks, 99% of the time is spent on I/Os, and 1% of the time is spent on CPUs. Therefore, improving the network transmission and read/write efficiency is the top priority.

Computing-intensive task: A large amount of computing is required, heavily consuming CPU resources. For example, the computing capability of the CPU can be used to calculate the circumference and decode HD videos. Computing-intensive tasks mainly consume CPU resources. Therefore, the code running efficiency is critical.

Data-intensive task: A large number of independent data analysis and processing jobs can be distributed on different nodes of a loosely coupled computer cluster system. The task involves mass data I/O throughputs with high density. In addition, most data-intensive applications have a process driven by data flows. Typical applications include log analysis, software as a service (SaaS), and business intelligence (BI) applications of large enterprises.

The computing modes of big data applications include batch processing, stream processing computing, graph computing, and query and analysis computing. Batch processing is used to process large amounts of data in batches. Major technologies include MapReduce and Spark. Stream computing is used to process stream data in real time. The main technologies include Spark, Storm, Flink, Flume, and DStream. Graph computing is used to process large-scale graph structure data. The main technologies are GraphX, Gelly, Giraph, and PowerGraph. Query and analysis computing are used to store, manage, query, and analyze a large amount of data. The main technologies include Hive, Impala, Dremel, and Cassandra.

1.2.2 Hadoop Big Data Ecosystem

The Hadoop ecosystem is mostly designed for processing large amounts of data greater than the volume on a single-node. You can compare it to a kitchen that needs all kinds of cookware. Each piece of cookware can be used separately or together with others.

For big data, we need to store data first. Therefore, we have Hadoop Distributed File System (HDFS). HDFS is designed to store a large amount of data across hundreds of machines, and data is stored in a file system rather than multiple file systems. For example, if you want to obtain the data in **/hdfs/tmp/file1**, the actual data is stored in

different machines but you only cite a file path. As a user, you do not need to know which track and which sector the file is distributed to. HDFS manages the data for you.

After the data is saved, we need to process and analyze it. In this case, **MapReduce/Tez/Spark /Flink** is required. MapReduce is the first-generation compute engine, while Tez and Spark are second-generation compute engines. Flink is mainly used for real-time computing. Developers find it troublesome to develop program code for these compute engines. To simplify development and improve efficiency, a higher-level abstract language layer is designed to describe algorithms and data processing processes. For this, Pig and Hive are available. Pig describes MapReduce in a similar way to scripts, while Hive uses SQLs. Both Pig and Hive convert the scripts and SQLs into MapReduce programs and then the compute engines process data. The language of Hive is SQL-like. Therefore, Hive is easy to use and becomes the core component of the big data warehouse.

In Hadoop 1.0, only MapReduce is used as the compute engine responsible for resource and job scheduling. In later versions, with the joining of other compute engines, job scheduling conflicts because a system can only schedule the resources and jobs by itself. To settle the conflict, Yarn and Oozie join the compute engine family. Yarn is responsible for system resource scheduling and management, and Oozie is responsible for compute job flow scheduling.

HDFS is the default persistent storage layer. HBase is a column-oriented distributed database and applies to structured storage. However, the bottom layer of HBase still depends on HDFS for physical storage, which is similar to Hive. However, Hive is suitable for analyzing and querying data in a period of time, and HBase is suitable for querying big data in real time. In addition, ZooKeeper is required for HBase deployment. ZooKeeper is a distributed coordination service, including the configuration service, metadata maintenance service, and NameSpace service.

Sqoop and Flume are developed to make up the gap where traditional data collection tools cannot fetch mass data. Sqoop is an open-source tool used to transfer data between Hadoop (Hive) and traditional databases (MySQL and PostgreSQL). You can import data from a relational database to HDFS of Hadoop or from HDFS to a relational database. Flume is a highly available, reliable, and distributed system provided by Cloudera for collecting, aggregating, and transmitting mass logs. Flume supports customization of various data senders in the log system to collect data.

In addition, there are some special systems and components. For example, Mahout is a distributed machine learning library, and Ambari is a distributed architecture software used to create, manage, and monitor Hadoop clusters.

In short, you can think of the big data ecosystem as a kitchen ecosystem. In order to cook different dishes, you need a variety of cookware. At the same time, customers' requirements are also keeping up with the times. Therefore, your tools need to be upgraded. In addition, no universal tool can handle all situations. As a result, the ecosystem will become larger and larger.

1.3 Challenges and Opportunities Faced by Enterprises

With the popularization of the Internet and the continuous development of information technologies, more data will be generated by society. From 2018 to 2025, the global datasphere will increase by more than five times. IDC predicts that the global datasphere will increase from 33 ZB in 2018 to 175 ZB in 2025. This requires more powerful big data storage, processing, and analysis capabilities. Traditional data processing methods have faced great challenges and brought about a series of potential problems, such as high storage costs of mass data, insufficient batch data processing performance, lack of stream data processing, limited scalability, and value-added data assets. This gives rise to the objectives of big data.

1.3.1 Big Data Challenges

Currently, technologies in the big data era are becoming mature. However, enterprises face some challenges when using these technologies:

- Unclear big data requirements for services

  Many enterprise business departments do not understand big data, and its application scenarios and value. Therefore, it is difficult for them to raise accurate requirements for big data. Because the requirements of business departments are not clear and big data departments do not generate revenue directly, the enterprise decision-makers are worried about the low input-output ratio. They are hesitant to set up big data departments, or even delete a large amount of valuable historical data due to the lack of application scenarios.

- Severe data silos in enterprises

  Data fragmentation is the most important challenge for enterprises starting to handle big data. In a large enterprise, different types of data are scattered across different departments. As a result, data in the same enterprise cannot be shared, and the value of big data cannot be brought into full play.

- Low data availability and quality

  Many large- and medium-sized enterprises generate a large amount of data every day, but they do not pay much attention to the preprocessing phase of big data. As a result, data processing is not standard. In the big data preprocessing phase, valid data is obtained after extraction, conversion, cleaning and noise reduction. Sybase data shows that the availability of high-quality data is improved by 10%, and enterprise profits are improved by more than 20%.

- Data-related management technologies and architecture

  Traditional databases are not suitable for processing PB-level data. Traditional databases do not consider the diversity of data, especially the compatibility of structured data, semi-structured data, and unstructured data. The O&M of mass data requires data stability to support high concurrency and reduce server load.

- Data security

  The wide use of internet makes it easier for criminals to get information about people, and gives rise to more means that are difficult to track and prevent. How to ensure user information security becomes an important issue in the era of big data. In addition, the increasing amount of big data poses higher requirements on physical

security of data storage, and therefore poses higher requirements on multiple data replicas and disaster recovery mechanisms.

- Lack of big data talents

  Each operation of component construction and maintenance in big data needs professional personnel. Therefore, it is required to build and cultivate a professional team that is experienced in big data management and applications. Every year, hundreds of thousands of big data-related jobs are created around the world. In the future, there will be a talent gap of more than 1 million workers. Therefore, universities and enterprises must work together to cultivate and explore talents.

- Trade-off between data openness and privacy

  As big data applications become increasingly important, data resource openness and sharing have become the key to maintaining advantages in the data war. However, data openness will inevitably infringe on some users' privacy. How to effectively protect citizens' and enterprises' privacy while promoting data openness, application, and sharing and gradually strengthen privacy legislation will be a major challenge in the big data era.

1.3.2 Big Data Opportunities

In addition to challenges, we need to pay more attention to opportunities brought by big data. In today's big data era, the business ecosystem has undergone great changes. The boundary between netizens and consumers is blurring. Ubiquitous smart terminals and online network transmission are required, and interactive social networks have made the profiles of people who used to be just web-browsers clear. For the first time, companies have the opportunity to conduct large-scale and precise consumer behavior research. They need to embrace this change proactively and adapt to this new era through self-transformation and evolution. The big data market will be hotly contested.

- Big data analysis becomes the core of big data technologies.

  An important prerequisite for realizing the value of big data is to eliminate false data, find out rules, and find valuable information from mass data. This cannot be achieved only by relying on expert experience and wisdom Various data mining technologies are required. The attitude of large enterprises towards data operations strategies varies for different companies. However, there is a consensus that we attach great importance to the construction of data analysis and mining capabilities and maximize the value of data through sustainable and in-depth data mining.

- Big data supports information technology applications.

  Next-generation information technologies, such as the mobile Internet, Internet of Things (IoT), social networks, digital homes, and e-commerce, use big data to aggregate generated information. These technologies process, analyze, and optimize data from different sources in a unified and comprehensive manner, and results are returned to various applications to further improve user experience and create huge business value, economic value, and social value. Therefore, big data has the power to accelerate social transformation. However, to unleash this power, we need strict data governance, insightful data analysis, and an environment that stimulates management innovation.

- Big data drives continuous growth of the information industry.

As the value of big data gains wider recognition among industry users, the market demand will surge, and new technologies, products, services, and business forms oriented to the big data market will emerge continuously. Big data will create a high-growth market for the information industry. In the hardware and integrated device field, big data will face challenges such as effective storage, fast read/write, and real-time analysis, which will have a significant impact on the chip and storage industries. In addition, integrated data storage and processing servers and in-memory compute markets are emerging. In the software and service field, the huge value of big data brings urgent requirements for quick data processing and analysis, which will lead to unprecedented prosperity in the data mining and business intelligence markets.

1.4 Huawei Kunpeng Solution

1.4.1 Introduction to Kunpeng

The current computing industry has two major trends: 1. Mobile smart terminals replace traditional PCs. The global shipment of traditional PCs in 2018 was 250 million, declining for the seventh consecutive year. In contrast, the global shipment of smart mobile terminals exceeded 1.6 billion in 2018. As a result, computing and applications are gradually transforming to mobile applications. This also brings new compute power requirements on the cloud data centers and devices. 2. Researches show that the world is entering an era where all things are connected. The number of connected devices worldwide exceeded 23 billion in 2018. By 2025, this figure is expected to exceed 100 billion, bringing massive quantities of data. In this era, the real-time intelligent processing capabilities on the edge side must be enhanced. The current AI compute power is sufficient for this. However, the analysis, processing, and storage of mass data at the data center side require high concurrency, high performance, and high throughput. These new requirements call for new compute power. In the future, there will be a huge amount of information, ubiquitous computing, and various computing application scenarios, from robotic vacuum cleaners to smart phones, smart homes, IoT, and smart driving. The diversity of scenarios leads to the diversity of data, including structured and unstructured data, such as digits, texts, images, videos, and images. Integer computing and floating-point computing are mainly applied in traditional computing. Integer computing has advantages in text processing, storage, and big data. Floating-point computing focuses on scientific computing. However, the traditional computing architecture cannot meet the requirements of data analysis and processing in different scenarios for various data types. A new computing architecture is required to solve this problem.

Against this background, Huawei initiated the Kunpeng computing industry. Under the framework of the Kunpeng computing industry, upstream and downstream vendors of the industry chain can develop products and solutions with differentiated advantages based on the existing Kunpeng and Arm ecosystems. As a member of the Kunpeng computing industry, Huawei has mastered the key technologies of the Arm64 processor core, micro-architecture, and chip design, has permanent authorization on the Armv8 architecture, and has developed the Kunpeng series processors. With long-term

investment and continuous innovation in the chip field, Huawei has built a differentiated competitive chip system covering computing, storage, transmission, management, and AI. Therefore, Huawei is capable of providing a computing foundation that supports the sustainable development of the Kunpeng computing industry. With the development of the Kunpeng computing industry in the Chinese market and the joining of more enterprises in China and abroad, the Kunpeng computing industry will eventually become a computing industry with continuous innovation capabilities and global leadership.

The Kunpeng computing industry is a collection of full-stack IT infrastructure, industry applications, and services powered by Huawei Kunpeng processors. This industry includes personal computers (PCs), servers, storage devices, OSs, middleware, virtualization, databases, cloud services, industry applications, and consulting and management services. With the powerful compute power provided by Kunpeng processors, Kunpeng computing will play an important role in the digital transformation of various industries.

In terms of ecosystem implementation, Huawei Cloud has worked with industry-leading independent software vendors (ISVs) and System Integrators (SIs) to create many success stories in industries such as finance, energy, government and enterprise, transportation, and retail. At the beginning of 2019, Huawei Cloud worked with ChinaSoft International to smoothly migrate nine service systems to the cloud for the largest dairy product supplier in China in only four hours. As cloud migration is the key service of the enterprise, Huawei Cloud helped it migrate up to 68 hosts and 14 databases to the cloud. In March 2019, Huawei Cloud and its partner Jingying Shuzi and China Coal Research Institute (CCRI), jointly developed the Mine Brain solution, an industrial intelligent twins solution for the coal industry.

The full openness to developers accelerates the implementation of Kunpeng industry cases. Huawei Cloud DevCloud has provided services for 300,000+ developers and has been deployed in 30+ city campuses with developed software industries in China. In the future, the Kunpeng industry will continue to strengthen cooperation and construction in ecosystem fields, including technology ecosystem, developer ecosystem, community construction, cooperation with universities, industry ecosystem, and partner ecosystem, to continuously enhance full ecosystem vitality.

In terms of computing capabilities, the TaiShan server based on the Huawei Kunpeng processor fully demonstrates the advantages of efficient, secure, and reliable computing. In addition, the TaiShan server is an open platform and supports mainstream software and hardware in the industry. TaiShan 100 is a first-generation server based on the Kunpeng 916 processor and was launched in 2016. In 2019, Huawei launched the TaiShan 200 server based on the latest Kunpeng 920 processor, which is the main product in the market.

The Kunpeng product system includes: mainstream OSs such as CentOS, Ubuntu, NeoKylin, Debian, Huawei Euler OS, Kylin, SUSE, and Deepin; niche OSs such as Chinese OSs (Hunan Kylin, Linx, YMOS, Taishan Guoxin, BCLinux, and NeoShine); non-Chinese OSs, such as Red Hat, a mainstream product outside China. It is temporarily removed from the market because it is subject to Export Administration Regulations (EAR). The Kunpeng product system is compatible with Red Hat. Currently, the Kunpeng processor supports only the Linux operating system.

In terms of cloud service applications, Huawei Cloud provides 69 Kunpeng cloud services (such as Kunpeng ECS, Kunpeng BMS, Kunpeng CCE, and Kunpeng CCI) and more than

20 solutions (such as Kunpeng DeC, Kunpeng HPC, Kunpeng big data, Kunpeng enterprise applications, and Kunpeng native applications) for governments, finance, enterprises, Internet, and other industries. Huawei Cloud Kunpeng cloud services and solutions have a full-stack ecosystem. Based on the Kunpeng community, Huawei Kunpeng Solution provides support for building mainstream components that cover multiple service scenarios and provides a platform for technical communication and discussion; completes the adaptation and compatibility certification of multiple open-source and operating systems, databases, middleware, and other system software in China; works with industry partners to develop industry-oriented Kunpeng solutions to serve end users.

1.4.2 Kunpeng Big Data Solution

On August 27, 2019, Huawei Cloud released the industry's first Kunpeng big data solution, Big Data Pro. This solution adopts the public cloud architecture with decoupled storage and computing, ultimate scalability, and the highest possible efficiency. The highly elastic Kunpeng compute power is used as the computing pool and and the OBS that offers ultimate data durability is used as the storage pool. The resource utilization of big data clusters can be greatly improved, and the big data cost can be halved. Big data services and AI training and inference platform, which feature high performance and reliability, enable quick digitalization and intelligent transformation.

This solution provides security, performance, efficiency, and openness.

- Security: Servers and big data platforms are independent and controllable. Chip-level data encryption ensures data confidentiality.
- Performance: The performance is improved by 30% compared with common servers of the same level. The solution delivers powerful compute capability and supports high-concurrency application scenario optimization. More than 5,000 nodes can be deployed in a big data cluster.
- Energy efficiency: Servers in this solution consume 30% less energy than general-purpose servers. With the same compute power, the rack space is reduced by 30%.
- Openness: This solution is compatible with the Arm ecosystem and supports mainstream hardware and software. OpenLab provides services such as software development, application porting, and compatibility certification.

1.4.3 Huawei Cloud Big Data Services

Using the one-stop big data platform MRS, Huawei Cloud big data service interconnects with DAYU and Data Lake Visualization (DLV) services to provide a unified platform for data integration, data governance, and data development to seamlessly connect to data foundations such as MapReduce Service (MRS), Data Warehouse Service (DWS), and Data Lake Insight (DLI) on Huawei Cloud. Enterprises can build a unified big data platform for data integration, storage, analysis, and mining in one-click mode, enabling them to focus on industry applications and improving development efficiency.

As a one-stop big data platform, MRS provides a unified O&M management platform where a cluster can be deployed with a few clicks. Scaling-out/Scaling-in and auto scaling can be done on a running cluster without interrupting services. Jobs and resource tags can be managed. MRS also provides one-stop O&M capabilities, such as monitoring, alarm reporting, configuration, and patch upgrade.

The data access layer provides data collection capabilities delivered by the components, including Flume (data ingestion), Loader (relational data import), and Kafka (highly reliable message queue). Data from various data sources can be collected. In the data analysis phase, you can select the Hive (data warehouse), SparkSQL, or Presto (both are interactive query engines) to analyze SQL-like data. MRS also provides multiple mainstream compute engines, including MapReduce (batch processing), Tez (DAG model), Spark (in-memory computing), SparkStreaming (micro-batch stream computing), Storm (stream computing), and Flink (stream computing) for various big data application scenarios. Both HDFS (a universal distributed file system) and OBS (featuring high availability and low cost) are used for underlying data storage.

MRS can connect to DAYU to provide one-stop data asset management, development, exploration, and sharing capabilities based on the enterprise data lake, helping users quickly build big data processing centers and implement data governance and development scheduling. This enables quick monetization of data.

Huawei Cloud big data services have the following advantages: 1. 100% compatibility with open-source ecosystems, plug-in management of third-party components, and a one-stop enterprise platform; 2. Decoupled storage and compute resources allowing for flexible configuration of storage and compute resources; 3. Application of Huawei-developed Kunpeng server. Thanks to the multi-core performance advantages of Kunpeng processors and algorithm optimization of task scheduling on Huawei Cloud, the CPU has higher concurrency capability, which provides higher compute power for big data computing.

1.4.4 Huawei Cloud MRS

Big data is a huge challenge facing the Internet era as data grows explosively both in volume and diversity. Conventional data processing techniques, such as single-node storage and relational databases, are unable to keep up. To meet this challenge, the Apache Software Foundation (ASF) has launched an open-source Hadoop big data processing solution. Hadoop is an open-source distributed computing platform that can fully utilize compute and storage capabilities of clusters to process massive amounts of data. However, if enterprises deploy Hadoop by themselves, they will encounter many problems, such as high costs, long deployment periods, difficult maintenance, and inflexible use. To solve these problems, Huawei Cloud provides MRS for managing the Hadoop system. With MRS, you can deploy the Hadoop cluster in just one click. MRS provides enterprise-level big data clusters on the cloud. Tenants can fully control clusters and easily run big data components such as Hadoop, Spark, HBase, Kafka, and Storm. MRS is fully compatible with open-source APIs. It incorporates the advantages of Huawei Cloud compute, storage and big data industry experience to provide customers with a full-stack big data platform featuring high performance, low cost, flexibility, and ease-of-use. In addition, the platform can be customized based on service requirements to help enterprises quickly build a mass data processing system and discover new value points and business opportunities by analyzing and mining massive amounts of data in real time or in non-real time.

MRS has a powerful Hadoop kernel team and is deployed based on Huawei's enterprise-level FusionInsight big data platform. MRS provides multi-level SLA assurance. MRS has the following advantages:

- High performance

  MRS supoorts the in-house CarbonData, a high-performance big data storage solution. It allows one data set to apply to multiple scenarios and supports features such as multi-level indexing, dictionary encoding, pre-aggregation, dynamic partitioning, and quasi-real-time data query. This improves I/O scanning and computing performance and returns analysis results of tens of billions of data records in seconds. In addition, MRS supports the self-developed enhanced scheduler Superior, which breaks the scale bottleneck of a single cluster and is capable of scheduling over 10,000 nodes in a cluster.

- Low cost

  Based on diversified cloud infrastructure, MRS provides various computing and storage choices and supports storage-compute decoupling, delivering cost-effective mass data storage solutions. MRS supports auto scaling to address peak and off-peak service loads, releasing idle resources on the big data platform for customers. MRS clusters can be created and scaled out when you need them, and can be terminated or scaled in after you use them, minimizing cost.

- High security

  With Kerberos authentication, MRS provides role-based access control (RBAC) and sound audit functions. MRS is a one-stop big data platform that allows different physical isolation modes to be set up for customers in the public resource area and dedicated resource area of Huawei Cloud as well as Huawei Cloud Stack Online in the customer's equipment room. A cluster supports multiple logical tenants. Permission isolation enables the computing, storage, and table resources of the cluster to be divided based on tenants.

- Easy O&M

  MRS provides a visualized big data cluster management platform, improving O&M efficiency. MRS supports rolling patch upgrade and provides visualized patch release information and one-click patch installation without manual intervention, ensuring long-term stability of user clusters.

- High reliability

  MRS delivers high availability (HA) and real-time SMS and email notification on all nodes.

  Big data is ubiquitous in people's lives. Huawei Cloud MRS is suitable for processing big data in the sectors such as the Internet of things (IoT), e-commerce, finance, manufacturing, healthcare, energy, and government affairs. Typical big data application scenarios are as follows:

- Large-scale data analysis

  Large-scale data analysis is a major scenario in modern big data systems. Generally, an enterprise has multiple data sources. After accessing the data sources, the enterprise needs to perform ETL processing on data to form model-based data for each service module to analyze and sort out. This type of service has the following characteristics: 1. The service does not have high requirements on real-time execution, and the job execution time ranges from dozens of minutes to hours; 2. The data volume is huge; 3. Various data sources and formats exist; 4. Data processing usually consists of multiple tasks, and resources need to be planned in

detail. In the environmental protection industry, climate data is stored on OBS and periodically dumped into HDFS for batch analysis. 10 TB of climate data can be analyzed in 1 hour. In this scenario, MRS has the following advantages: 1. Low cost: OBS is used to implement low-cost storage; 2. Analysis of massive sets of data: TB/PB-level data is analyzed by Hive; 3. Visualized data import and export tool: Loader exports data to DWS for business intelligence (BI) analysis.

- Large-scale data storage

  A user who has a large amount of structured data usually requires index-based quasi-real-time query. For example, in an Internet of Vehicles (IoV) scenario, vehicle maintenance information is queried by plate numbers. Therefore, vehicle information is indexed by the plate number when stored, to implement second-level response in this scenario. Generally, the data volume is large, and the user may store data for one to three years.

  For example, in the IoV industry, an automobile company stores data on HBase, which supports PB-level storage and CDR queries in milliseconds. In this scenario, MRS has the following advantages: 1. Real-time: Kafka implements real-time access of messages from a large number of vehicles. 2. Storage of massive sets of data: HBase stores a large volume of data and implements millisecond-level data query. 3. Distributed data query: Spark analyzes and queries a large volume of data.

- Real-time data processing

  Real-time data processing is usually used in scenarios such as anomaly detection, fraud detection, rule-based alarming, and service process monitoring. Data is processed while being input to the system. For example, in the Internet of elevators & escalators (IoEE) industry, data of smart elevators and escalators is imported to MRS streaming clusters in real time for real-time alarming. In this scenario, MRS has the following advantages: 1. Real-time data collection: Flume collects data in real time and provide various collection and storage connection modes. 2. Data source access: Kafka accesses data of tens of thousands of elevators and escalators in real time.

1.5 Quiz

1. What challenges do we face in the big data era?

2 HDFS and ZooKeeper

2.1 HDFS: Distributed File System

2.1.1 HDFS Overview

The Hadoop distributed file system (HDFS) is designed to run on commodity hardware. HDFS has many similarities with existing distributed file systems, but at the same time, it is very different from other distributed file systems. HDFS is a highly fault-tolerant system that can be deployed on low-cost machines. It provides high-throughput data access and is suitable for applications with large-scale data sets. HDFS was initially developed as the infrastructure of the Apache Nutch search engine project. HDFS is part of the Apache Hadoop Core project.

HDFS is suitable for large-scale dataset storage and application. It can be used to store website data, electronic traffic eye data, meteorological data, and sensor data.

2.1.2 HDFS Concepts

HDFS uses a typical master/slave system architecture. An HDFS cluster usually contains one NameNode and multiple DataNodes. A file is divided into one or more data blocks and stored on a group of DataNodes. The DataNodes can be distributed on different racks. The NameNode opens, closes, and renames files or directories in the namespace of the file system, and manages the mapping between data blocks and DataNodes. Under the unified scheduling of NameNode, DataNodes process read/write requests from file system clients, and create, delete, and replicate data blocks.

- NameNode (management node): manages the master server of the file system namespace and the access of the management client to files. For example, you can open, close, and rename files and directories. NameNode manages the mapping between file directories, files, and blocks, and the mapping between blocks and DataNodes, maintains the directory tree, and takes over user requests. This information is stored in the local file system in two forms, one is namespace image, and the other is editing logs. You can obtain the location of each block of each file stored on the DataNode from the NameNode. The NameNode dynamically rebuilds the information each time the system is started. The client obtains metadata information through the NameNode and interacts with the DataNode to access the entire file system.
- DataNode: (data node): the working node of the file system, invoked by the client and NameNode to execute specific tasks and store file blocks. DataNodes periodically send file block information to NameNodes through the heartbeat mechanism to report their working status. DataNodes also create and delete blocks.

- Client: used for users to access the entire file system by interacting with NameNode and DataNode. HDFS opens file namespaces and allows user data to be stored as files. Users communicate with the HDFS through the client.
- Data block: the minimum unit data read or write on a disk. Files are stored on disks as blocks. The file system can process data blocks whose size is an integer multiple of the data block size on a disk each time. Files in the HDFS are also divided into multiple logical blocks for storage. In versions later than Hadoop 2.0, the default data block size is 128 MB. As a distributed file system, HDFS has the following advantages when using data blocks for storage:
    - Large-scale file storage: Files are stored in blocks. A large-scale file can be split into multiple file blocks, and different file blocks can be distributed to different nodes. Therefore, the size of a file is not limited by the storage capacity of a single node. The capacity can be much larger than the storage capacity of any node on the network.
    - Simplified system design: First, HDFS greatly simplifies storage management because the file block size is fixed. In this way, it is easy to calculate the number of file blocks that can be stored on a node. Second, HDFS facilitates metadata management. Metadata does not need to be stored together with file blocks. It can be managed by other systems.
    - Suitability for data backup: Each file block can be redundantly stored on multiple nodes, greatly improving the fault tolerance and availability of the system.

2.1.3 HDFS Key Features

2.1.3.1 High Availability

NameNode stores metadata information (FSImage) and operation logs (EditLog). Any read or write operation performed by the client on HDFS can be performed only after the metadata is obtained from NameNode. The health condition of NameNodes directly affects the use of the entire storage system. If NameNode is faulty, no clients can perform operations on data in HDFS.

Hadoop2.*x* supports HDFS high availability (HA), which solved the single-point NameNode faults. Figure 2-1 shows the HDFS HA architecture. In this architecture, ZooKeeper is used to implement active and standby NameNodes to ensure HDFS reliability. Two NameNodes work in redundancy mode, of which one is the active NameNode. Similar to the NameNode in non-HA mode, the active NameNode receives remote procedure call (RPC) requests from clients and provides services. The other NameNode in the standby state is the standby NameNode. It synchronizes metadata from the active NameNode and serves as the hot backup of the active NameNode. When the active NameNode is faulty, the standby NameNode quickly takes over the requests from the client to restore the HDFS service.

Hadoop 3.*X* allows users to run multiple standby NameNodes. However, there is only one active NameNode and the remaining NameNodes are standby ones. In this architecture, the cluster can tolerate the failure of multiple NameNodes.

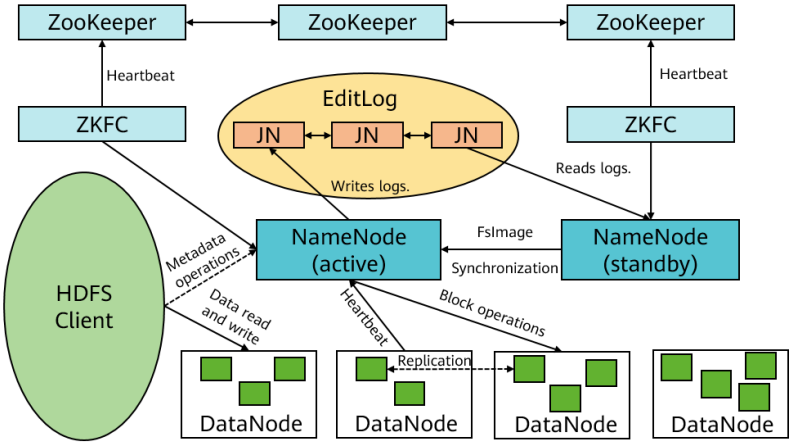


Figure 2-1 HDFS HA architecture

2.1.3.2 Metadata Persistence

HDFS metadata (description file) persistence consists of FSImage and EditLog files. EditLog records user operation logs and is used to generate new file system images based on FSImage. FSImage saves file images periodically.

Figure 2-2 shows the metadata persistency process when HDFS is continuously updated. Step 1: When HDFS is formatted for the first time, the NameNode (active NameNode in the figure) generates the FSImage and EditLog files. Step 2: The standby NameNode (standby NameNode in the figure) downloads FSImage from the active NameNode and reads EditLog from the shared storage. Step 3: The standby NameNode combines the logs and old metadata to generate new metadata **FSImage.ckpt**. Step 4: The standby NameNode uploads the metadata to the active NameNode. Step 5: The active NameNode rolls back the uploaded metadata. Finally, the first step is repeated.

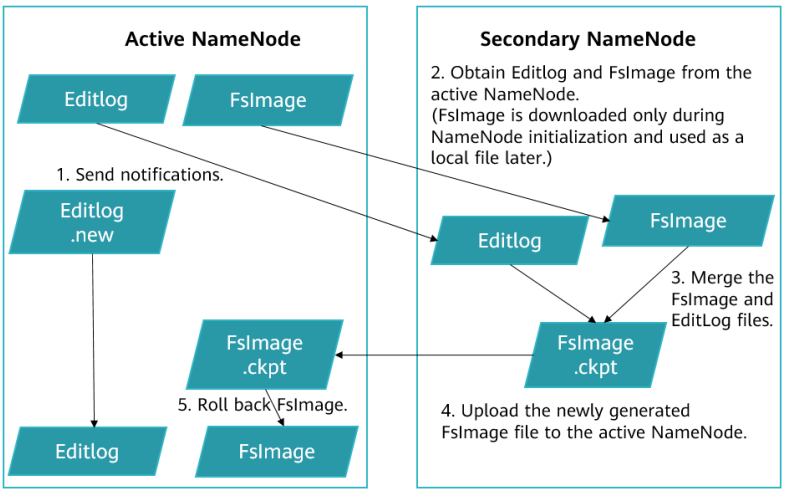


Figure 2-2 Metadata persistence process

2.1.3.3 HDFS Federation

The HDFS federation can be regarded simply as the aggregation of multiple HDFS clusters. More accurately, it is an HDFS cluster with multiple NameNodes. When the cluster size reaches to a certain degree, the memory used by a NameNode process may be more than 100 GB. In this case, the NameNode becomes the performance bottleneck.

In addition, the metadata maintained in a single active NameNode is in the same namespace, and applications in the cluster cannot be isolated. A program's manipulation of metadata may affect other running programs.

In the HDFS federation, there are multiple joint but independent NameNodes, so the HDFS namespace can be horizontally expanded. Each NameNode maintains its own namespace volume, including the metadata of the namespace and a block pool. The blockpool manages all data blocks of files in the namespace. The block pool allows a namespace to create a block ID for a new data block without notifying other NameNodes. In this way, when the DataNode reports data block information to all NameNodes in the cluster, each namespace manages only a group of data blocks that belong to it. Therefore, the service interruption of one NameNode does not affect the availability of data in other NameNodes. Each NameNode is responsible for only some directories in the entire HDFS cluster. Metadata is not shared among NameNodes. Each NameNode has a standby node.

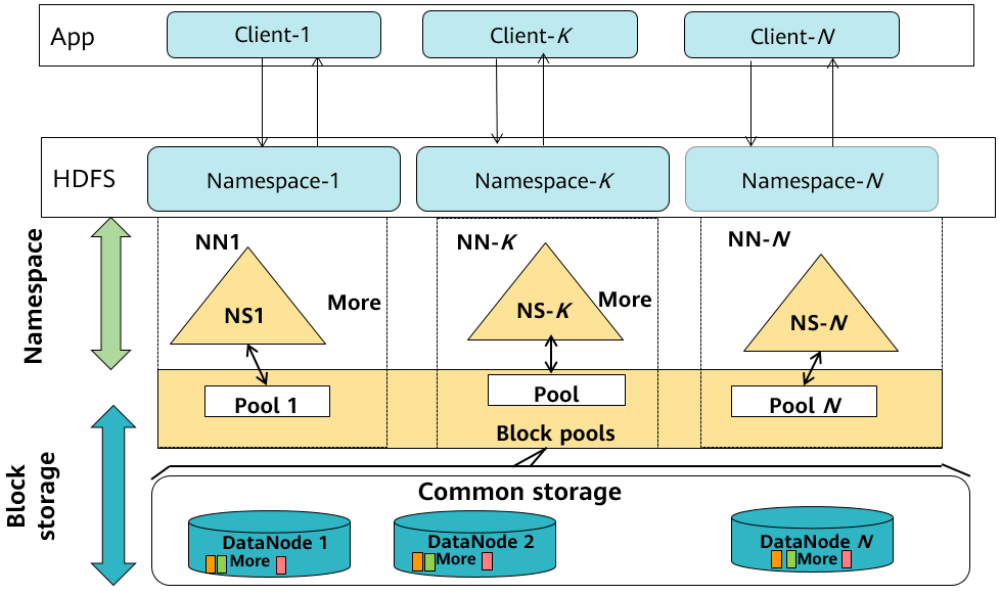


Figure 2-3 HDFS federation mechanism

The HDFS federation mechanism is mainly used for ultra-large file storage. For example, Internet companies store large-scale data such as user behavior data, historical telecom data, and audio data.

2.1.3.4 Data Replica Mechanism

HDFS stores replicas on different racks. As shown in Figure 2-4, the first replica B1 is on the local machine, the second replica B2 is on the remote rack, and the third replica B3 checks whether the two replicas are on the same rack. If so, another rack is selected. If no, a different node on the same rack as B1 is selected. For the fourth replica or other replicas, a storage location is randomly selected.

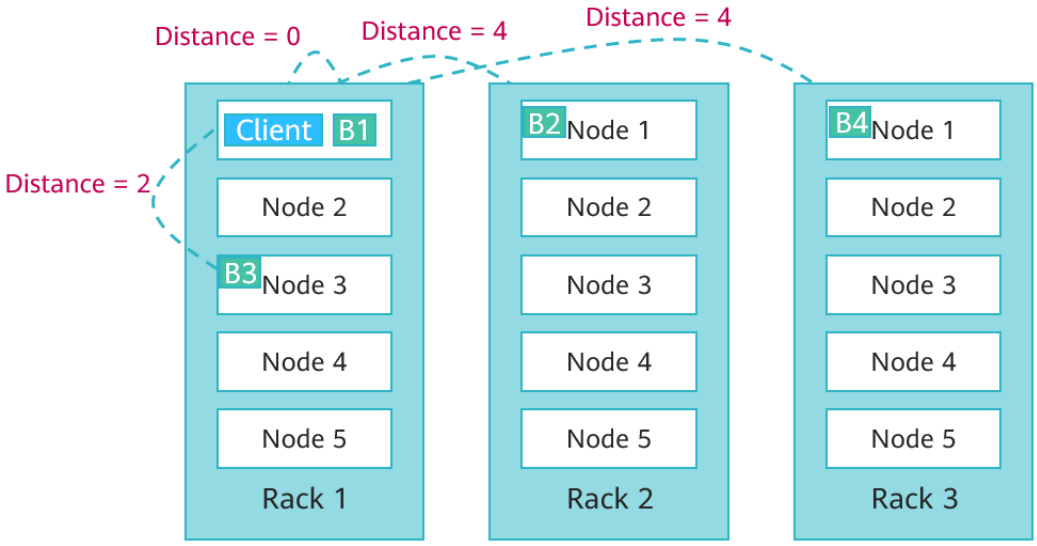


Figure 2-4 Data replica storage

The rack awareness policy of the HDFS system prevents data loss caused by single rack failure and allows the bandwidth of multiple racks to be fully utilized when data is read. In HDFS, the data replica nearest to the client is read preferentially to reduce the overall bandwidth consumption, thereby reducing the overall bandwidth latency.

HDFS uses the following formula to calculate the replica distance:

- Distance (Rack1/D1Rack1/D1) =0 #The distance between the same server is 0.
- Distance (Rack1/D1Rack1/D3) =2 #The distance between different servers on the same rack is 2.
- Distance (Rack1/D1Rack2/D1) =4 #The distance between different servers on different racks is 4.

Rack1 and Rack2 are cabinet IDs, and D1, D2, and D3 are the host IDs of DataNodes in the cabinet. That is, the distance between two data blocks of the same host is 0; the distance between two data blocks on different hosts of the same rack is 2; the distance between data blocks on hosts of different racks is 4.

Through rack awareness, HDFS in the working state always tries to ensure that at least two of the three replicas (or more replicas) of a data block are in the same rack, and at least one of the three replicas is in a different rack (at least two racks).

2.1.3.5 HDFS Data Integrity Guarantee

HDFS aims to ensure the integrity of stored data and ensures the data reliability in case of component failure.

- Rebuilding the data replica of failed data disks

  DataNode and NameNode periodically report data status through heartbeat messages. The NameNode manages whether blocks are completely reported. If the DataNode does not report data blocks because a hard disk is damaged, the NameNode initiates replica rebuilding to restore the lost replicas.

- Cluster data balancing

HDFS designs the data balancing mechanism, which ensures that data is evenly distributed on each DataNode.

- Metadata reliability

  Metadata can be operated using the log mechanism, and metadata is stored on the active and standby NameNodes.

  The snapshot mechanism implements the common snapshot mechanism of file systems, ensuring that data can be restored promptly in the case of misoperations.

- Security mode

  When a hard disk of a node is faulty, the node enters the security mode. In this mode, HDFS supports only access to metadata. In this case, data on HDFS is read-only. Other operations, such as creating and deleting files, will fail. After the disk fault is rectified and data is restored, the node exits the security mode.

New Features of HDFS 3.0

- Erasure encoding (EC) in HDFS

  EC is a data protection technology. It is the first data recovery technology used in data transmission in the communications industry, and is also a fault-tolerant coding technology. By adding new parity data to the original data, data in different parts is correlated. For a certain range of data errors, EC can be used to restore the data, which prevents data loss and solves the problem of doubling the HDFS storage space.

  When a file is created, the EC policy is inherited from the nearest ancestor directory to determine how its blocks are stored. Compared with 3-channel replication, the default EC policy can save 50% storage space and tolerate more storage faults.

  It is recommended that EC storage be used for cold data of large quantity. The number of replicas can be reduced to reduce the storage space. In addition, cold data is stable. Once the data needs to be restored, services are not greatly affected.

- Combination based on the HDFS router

  HDFS adds an RPC routing layer based on a federation of routers and provides a combined view of multiple HDFS namespaces. This is similar to the existing combination of ViewFS and HDFS. The difference is that the installation table is managed by the routing layer on the server side instead of the client side. This simplifies access to federated clusters on existing HDFS clients. This is similar to the existing combination of ViewFS and HDFS. The difference is that the installation table is managed by the routing layer on the server side instead of the client side. This simplifies access to federated clusters on existing HDFS clients.

- Multiple NameNodes

  Users can run multiple standby NameNodes. For example, by configuring three NameNodes and five JournalNodes, the cluster can tolerate the failure of two nodes instead of one. However, there is only one active NameNode, and the remaining NameNodes are standby ones. The standby NameNodes continue synchronizing with the JournalNodes to ensure that they obtain the latest EditLog files and synchronize the edits to the image maintained by themselves. In this way, hot backup can be implemented. When a failover occurs, the standby NameNode immediately switches

to the active state and provides services for external systems. In addition, the JournalNodes allows only one active NameNode to be written.

- Adding disk balancers to DataNodes

  A data balancer between different disks can be added on a single DataNode. The Hadoop of earlier versions supports balancers only between DataNodes. If data imbalance occurs between different disks on each node, there is no good method to handle the problem. You can run the **hdfs diskbalancer** command to balance data among disks on a node. This function is disabled by default. You need to manually set **dfs.disk.balancer.enabled** to **true** to enable it.

2.1.4 HDFS File Read and Write

2.1.4.1 File Read

- First, the client calls the **open()** function in the FileSystem object to open the target file and obtain a DistributedFileSystem instance.
- Then, the DistributedFileSystem invokes a NameNode by using a remote procedure call (RPC) to obtain location information of the first batch of file blocks, and obtain location information of the first batch of file blocks. The location of multiple DataNodes is returned for the same block based on the number of backups. The DataNodes are sorted based on the network topology of the cluster. The DataNode closest to the client is placed in the front. If the client is the DataNode, the client reads files from the local host.
- The DistributedFileSystem class returns an FSDataInputStream object to the client to read data. The object is encapsulated into a DFSInputStream object, which manages the I/O data flow between the DataNode and NameNode. The client calls the **read()** method on the input end, and DFSInputStream finds the DataNode nearest to the client and connects to the DataNode.
- The **read()** function is repeatedly called in the data flow until the block is completely read. DFSInputStream disconnects from the DataNode, and reads the next block. These operations are transparent to the client. From the perspective of the client, these operations are only reading a continuous stream.

  Each time a block is read, checksum verification is performed. If an error occurs when the DataNode is read, the client notifies the NameNode and continues to read data from the next DataNode that has the copy of the block.
- After the data of the current block is correctly read, the current DataNode connection is disabled and the optimal DataNode is found for reading the next block. If the first batch of blocks has been read and the file reading is not complete, DFSInputStream obtains the location of the next batch of blocks from NameNode and continues the reading.
- After the file reading ends on the client, the **close** method of FSDataInputStream is called to close all streams.

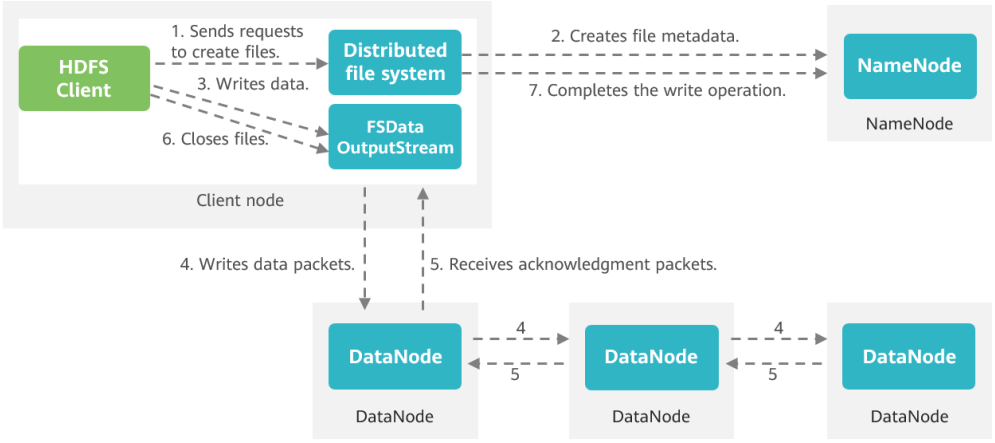


Figure 2-5 Process for a client to read data from HDFS

2.1.4.2 File Write

- The client calls the **create()** function on the DistributedFileSystem object to create a file.
- DistributedFileSystem sends a request to a NameNode through RPC to create a file that is not associated with a block. Before the file is created, the NameNode performs various checks, for example, checking whether the file exists and whether the client has the permission to create the file. If the verification is successful, the NameNode records the information and returns the DataNode address information corresponding to the block list (all replicas) of the file. Otherwise, an I/O exception is thrown.
- DistributedFileSystem returns the object of FSDataOutputStream for the client to write data. FSDataOutputStream is encapsulated into DFSOutputStream, which can coordinate NameNode and DataNode. The client starts to write data to DFSOutputStream. DFSOutputStream divides the data into packets and manages the packets in the form of data queues.
- DataStreamer receives and processes the data queue, applies for blocks from NameNode, obtains the DataNode list for storing replicas, and arranges them into a pipeline (the list size is determined by the replication setting in NameNodes). The DataStreamer outputs packets to the first DataNode in the pipeline in sequence, stores the packets, and then transfers the packets to the next DataNode in the pipeline up to the last DataNode. This data write mode is in the form of a pipeline.
- After the last DataNode is successfully stored, an ack packet (acknowledgment queue) consisting of packets (storing the sent data blocks) is returned. The DataStreamer updates all data blocks to the DataNodes in the pipeline and waits for the ack queue to return. After receiving the ack packets from the DataNodes, the client notifies the DataNodes to mark the file as completed and removes the corresponding packets from the ack queue.
- After the data is successfully written, the client invokes the **close()** function for the data flow. This invocation writes all the remaining data packets to the DataNode pipe, and the client connects to the NameNode, and waits for the confirmation message.

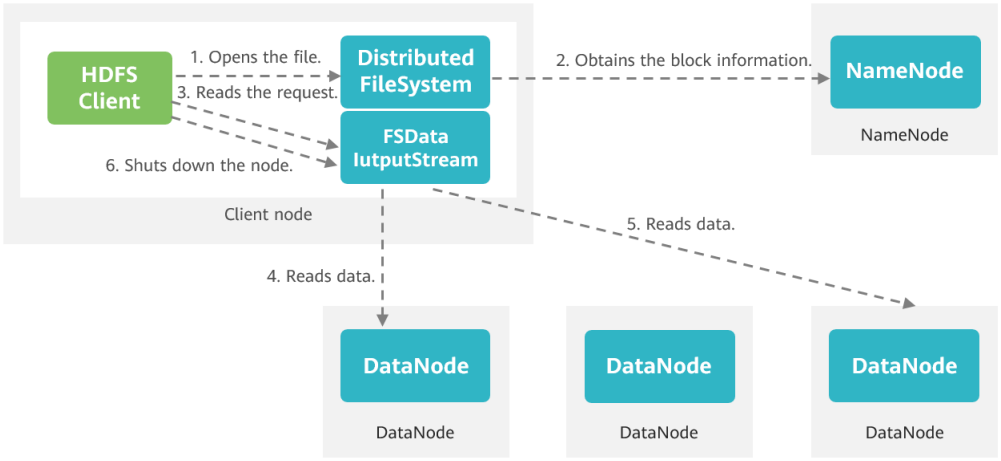


Figure 2-6 Process for a client writing data to HDFS

2.2 ZooKeeper: Distributed Coordination Service

2.2.1 ZooKeeper Overview

ZooKeeper is a typical distributed data consistency solution. Using ZooKeeper, distributed applications can implement functions such as data publication/subscription, load balancing, naming service, distributed coordination/notification, cluster management, master election, distributed lock, and distributed queue. ZooKeeper encapsulates complex and error-prone key services and provides users with easy-to-use APIs and efficient and stable systems. In security mode, ZooKeeper depends on Kerberos and LdapServer for security authentication. One of the most common application scenarios of ZooKeeper is a registry for service producers and consumers.

2.2.2 ZooKeeper Architecture

2.2.2.1 Introduction

A ZooKeeper cluster consists of a group of servers, including one leader node and multiple follower nodes. When a client is connected to the ZooKeeper cluster and initiates a write request, the request is sent to the leader node. Then data changes on the leader node are synchronized to the follower nodes in the cluster.

After receiving a data change request, the leader node first writes the change to local disks for restoration. All data changes are stored in memory only after being written to disks.

ZooKeeper uses the custom atomic message protocol to ensure data or status consistency among nodes in the entire coordination system. Followers use this protocol to synchronize local ZooKeeper data to the leader node and provide services based on local storage.

If the leader node is faulty, the message layer selects another leader node (when a server node obtains more than half of the votes, it becomes the new leader) to process the write requests of clients and synchronize data changes to the ZooKeeper coordination system to other follower nodes.

For a service with *n* instances, *n* may be an odd or even number. Assume that the DR capability is *x*. If the value is an odd number and *n* is 2x + 1, x + 1 votes are required for a node to become the leader.

If the value is an even number and *n* is 2x + 2, x + 2 votes are required for a node to become the leader.

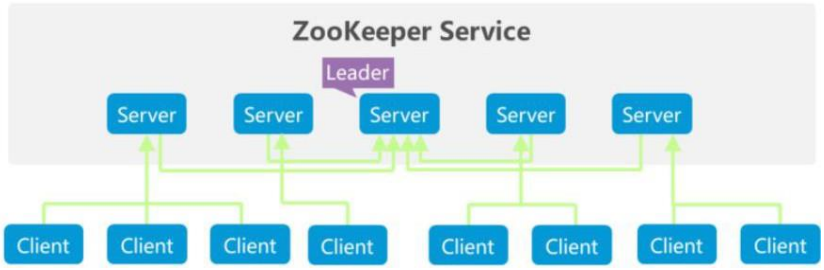


Figure 2-7 ZooKeeper service architecture

2.2.2.2 ZooKeeper Read/Write Mechanism

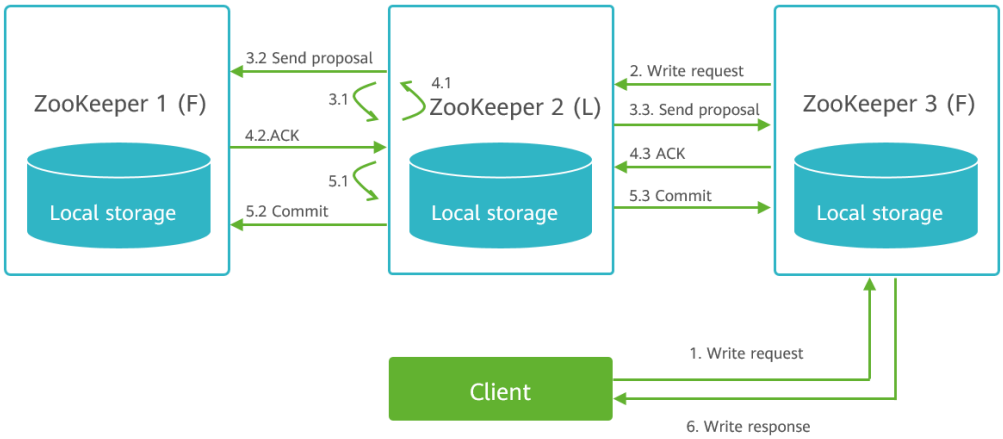


Figure 2-8 ZooKeeper write process

- A client can send a write request to any server.
- The server sends this request to the leader.
- After receiving the write request, the leader sends the write request to all nodes and confirm whether the write operation can be performed.
- The followers send ACK messages. The leader determines whether the write operation can be performed based on the ACK messages. If the number of write operations that can be performed is greater than 50% of the total instances, the write operation can be performed.
- The leader sends the result to the followers and performs the write operation. The leader data is synchronized to the followers. The write operation is complete.

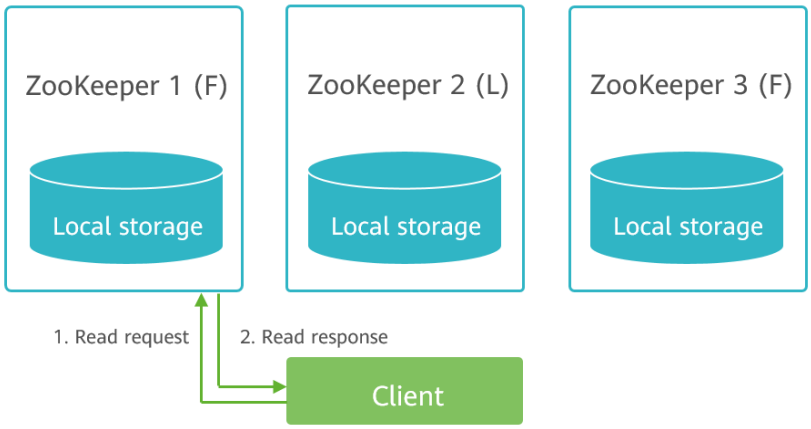


Figure 2-9 ZooKeeper read process

- A client can send a write request to any server.
- No matter which server the client is connected to, the same view is displayed to the client. The server directly returns the result.

2.2.2.3 Key Features of ZooKeeper

ZooKeeper is everywhere in the big data framework. The features of ZooKeeper determine the service performance factors, such as high availability. ZooKeeper has the following characteristics:

- Eventual consistency: No matter which server the client is connected to, the client is displayed in the same view. This is the most important performance of ZooKeeper.
- Reliability: ZooKeeper features simple, robust, and excellent performance. If message m is received by one server, it will be received by all servers.
- Real-time capability: Clients can obtain server updates and failures within a specified period of time. However, due to reasons such as network delay, ZooKeeper cannot ensure that two clients obtain the updated data at the same time. If you need the latest data, invoke the **sync()** API before reading the data.
- wait-free: Slow or faulty clients cannot intervene in the requests of rapid clients, so the requests of each client can be processed effectively.
- Atomicity: Data transfer either succeeds or fails. There is no middle status.
- Sequence: global sequence and partial sequence. Global sequence means that if message A is published before message B on one server, the same sequence applies to all servers. Partial sequence means that if message B is published after message A by the same sender, message B must come after message A.

2.2.2.4 ZooKeeper CLI-based Operations

The ZooKeeper command line (CLI) is simple. After ZooKeeper is installed, you can use built-in script **zkCli.sh** to connect to the ZooKeeper server. The common ZooKeeper operations are as follows:

- To create a node: create /node
- To list subnodes: ls /node
- To create node data: set /nodedata

- To obtain node data: get /node
- To delete a node: delete /node

2.3 Quiz

1. Why is the size of an HDFS data block larger than that of a disk block?
2. Can HDFS data be read when it is written?

3 HBase and Hive

3.1 HBase: Distributed Database

3.1.1 HBase Overview and Data Models

3.1.1.1 HBase Overview

HBase, short for Hadoop Database, is a distributed column-oriented database built on the Hadoop file system (HDFS). It provides fast random access to massive amounts of structured data. As a part of the Hadoop file system, HBase provides random real-time read/write access to data, the fault tolerance capability provided by HDFS is used. HBase is a distributed database solution that uses a large number of inexpensive machines to store and read massive data at a high speed.

HBase is an open-source, non-relational, and distributed database (NoSQL database). It references Google's BigTable modeling and uses Java as the programming language. As a part of the Hadoop project of the Apache Software Foundation, it runs on the HDFS file system and provides services similar to BigTable services for Hadoop. Therefore, it provides a high fault tolerance rate for sparse files and is suitable for storing big table data.

HBase implements the compression algorithm, memory operations, and Bloom filters on columns, which are described in the BigTable thesis. HBase tables can be used as the input and output of MapReduce tasks. They can be archived and backed up on the Java API page, stored in Internet Archive, or accessed through REST, Avro, or Thrift APIs.

Although the performance has been greatly improved recently, HBase cannot directly replace the SQL database. Today, it has been applied to multiple data-driven websites, including Facebook's messaging platform.

HBase is a highly reliable, column-oriented, and scalable distributed storage system that delivers excellent performance. HDFS provides it with reliable underlying data storage services; MapReduce provides it with high-performance computing capabilities; ZooKeeper provides it with stable services and a failover mechanism. ZooKeeper has the following functions: distributed lock, event monitoring, data storage of HBase on RegionServer, which functions as a micro database.

3.1.1.2 Comparison Between HBase and RDB

HBase differs from traditional relational databases in the following aspects:

- Data indexing: A relational database can build multiple complex indexes for different columns to improve data access performance. HBase has only one index, that is, the row key. All access methods in HBase can be accessed using the row key or row key scanning. This ensures proper system running.

- Data maintenance: In the relational databases, the most recent value replaces the original value in the record. The original value no longer exists after being overwritten. When an update is performed in HBase, a new version is generated, and the original one is retained.
- Scalability: It is difficult to horizontally expand relational databases, and the space for vertical expansion is limited. In contrast, distributed databases, such as HBase and BigTable, are developed to implement flexible horizontal expansion. Their performance can easily be scaled by adding or reducing the hardware in a cluster.

3.1.1.3 HBase Application Scenarios

Since HBase was developed nearly 10 years ago, HBase has become a mature project of the Apache Foundation. Many companies, such as Cloudera, support different branch versions. Now, let's learn about some specific applications.

- Object storage: Data on the Internet such as documents, web pages, and images in headline and news apps is stored in HBase. Some antivirus companies store their virus libraries in HBase.
- Time series data: The OpenTSDB module of HBase can be used to meet the processing requirements in time series scenarios to process data such as sensor data, monitoring data, and Candlestick chart data.
- Recommendation profile: User profiles and commodity profiles are large sparse matrices, which are suitable for HBase storage. For example, Ant Group's risk control data is stored in HBase.
- Meteorological data: This type of data mainly includes WeChat track data and meteorological data. Most of this data is stored in HBase.
- CubeDB analysis: Kylin is a cube analysis tool. Data at the bottom-layer is stored in HBase. Many customers build cubes using offline computing and store the cubes in HBase to meet online report query requirements.
- Message/Order: This type of data includes user behavior data, such as the Taobao favorites, transaction data, and Wangwang chat records of Alibaba. The data is directly stored in HBase.
- Feeds: One of the typical applications is Wechat Moments.
- NewSQL: With the Phoenix plug-in, NewSQL can meet the requirements of secondary indexes and SQLs as well as SQL non-transactional requirements for interconnecting with traditional data.

3.1.1.4 HBase Data Model

Applications store data in HBase as tables. Each table has rows and columns. All columns belong to a column family.

The table structure is sparse. The intersection of rows and columns is called a cell, and cells are versioned. The content of a cell is an inseparable byte array.

A rowkey of a table is also a byte array, so anything can be saved, whether it is a string or number.

All tables must have a primary key. HBase tables are sorted by key, and the sorting mode is based on byte.

Row Key	column-family1		column-family2			column-family3	...
	column1	column2	column1	column2	column3	column1	
Key1	t1:abc T2:gdfx			t4:hello t3:world			
Key2		t2:xxzz t1:yyxx					

Figure 3-1

3.1.1.5 HBase Table Structure

In HBase, the data model also consists of tables. Each table has rows and columns to store HBase data. The following figure shows the logical structure of HBase.

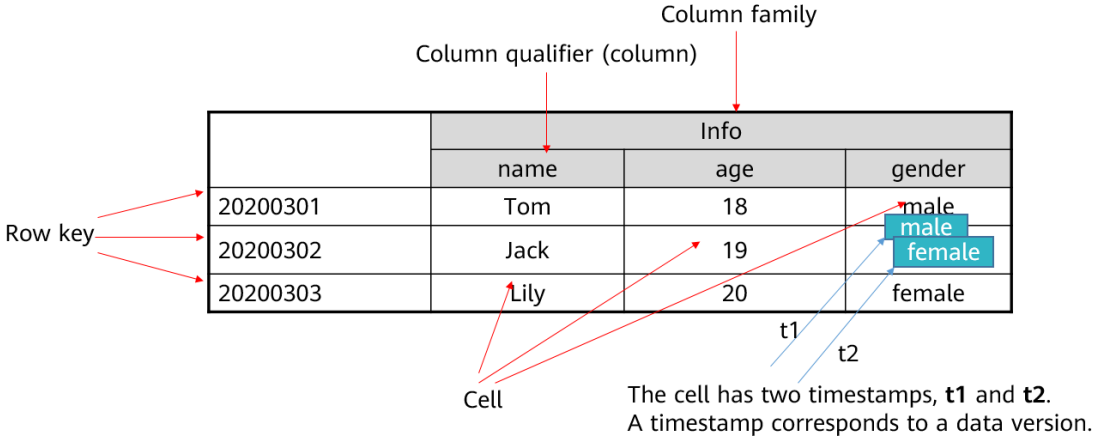


Figure 3-2 Logical structure of an HBase data table

Introduction to HBase Data Model

Table: HBase organizes data into a table, that is, an HBase table containing multiple rows.

Row: A row in HBase contains a row key and one or more columns related to the key value. Rows are sorted in alphabetical order when they are stored. For this reason, the design of row keys is very important. The objective is to store data in a way that related rows are close to each other. Website domain is a common row key pattern. If your row keys are domain names, you might want to store them in the opposite location (org.apache.www, org.apache.mail, org.apache.jira). In this way, all Apache domains in the table are close to each other, rather than being distributed according to the first letter of the subdomain.

Column: Columns in HBase consists column families and column qualifiers which are separated by colons (:).

Column family: For performance reasons, a column family physically has a group of columns and their values. In HBase, each column family has a set of storage attributes, for example, whether its value should be cached in memory, how data is compressed, and how its row code is encoded. Each row in the table has the same column family, but a given row may not store anything in the given column family. Once a column family is determined, it cannot be easily modified because it affects the actual physical storage structure of HBase. However, the column qualifier and its values in the column family can be dynamically added or deleted.

Column qualifier: A column qualifier is added to a column family to provide an index for a given data segment. About the content of the column family, the column qualifier might be content:html, and the other might be content:pdf. Although the column family is fixed when a table is created, the column qualifiers are variable and may vary greatly between rows.

Cell: A cell is a combination of row, column family, and column qualifiers. It contains values and timestamps which indicate the version of a value.

Timestamp: A timestamp is written with each value and is the identifier of the value of a given version. By default, the timestamp indicates the time on RegionServer when data is written. You can specify different timestamp values when placing data in cells.

Compared with traditional data tables, HBase data tables have the following characteristics:

- Each table has rows and columns. All columns belong to a column family.
- The intersection of a row and a column is called cell, and the cell is versioned. The content of a cell is an inseparable byte array.
- The row key of a table is also a byte array, so any data can be saved, whether it is a string or a number.
- HBase tables are sorted by key, and the sorting mode is based on byte.
- All tables must have primary keys.

3.1.1.6 HBase Concept View

The example in this section has a table named **webtable** that contains two rows (**com.cnn.www** and **com.example.www**) and three column families named **contents**, **anchor**, and **people**. In this example, for the first row (**com.cnn.www**), **anchor** contains two columns (**anchor:cssnsi.com**, **anchor:my.look.ca**), and **contents** contains one column (**contents:html**). This example contains five versions of a row with the row key **com.cnn.www** and one version of a row with the row key **com.example.www**. The column qualifier **contents: html** contains the entire HTML of a given website. Each qualifier of the **anchor** column family contains the external site that links to the site represented by the row, along with the text it used in the anchor of its links. The column family **people** represents people associated with the site.

A column name consists of the column family prefix and qualifier. For example, the column content **html** consists of the column family **contents** and qualifier **html**. Colon (:) separates column families from column family qualifiers.

The **webtable** table is as follows:

3.1.1.7 HBase Physical Views

Although tables are considered as collections of sparse rows in the HBase conceptual view, they are physically stored by column family. You can add a new column qualifier (column_family: column_qualifier) to an existing column family at any time.

Table 3-1 ColumnFamilyanchor

Row Key	Timestamp	ColumnFamily Anchor
"com.cnn.www"	T9	anchor:cnnsi.com="CNN"

"com.cnn.www"	T8	anchor:my.look.ca="CNN.com"

Table 3-2 ColumnFamily Content

RowKey	Timestamp	ColumnFamily Content
"com.cnn.www"	T6	contents:html="<html>..."
"com.cnn.www"	T5	contents:html="<html>..."
"com.cnn.www"	T3	contents:html="<html>..."

Empty cells shown in the HBase conceptual view are not stored at all. Therefore, no value is returned for the requests whose timestamp is **t8** in the **contents:html** column. Similarly, no value is returned for a request with the **anchor:my.look.ca** value in timestamp **t9**. However, if no timestamp is provided, the latest value of a specific column is returned. Given multiple versions, the most recent one is also the first one because the timestamps are stored in descending order. Therefore, if no timestamp is specified, the request for the values of all columns in row **com.cnn.www** will be **contents:html** in timestamp **t6**, **anchor:cnnsi.com** in timestamp **t9**, and **anchor:my.look.ca** in timestamp **t8**.

3.1.2 HBase Architecture

3.1.2.1 Introduction to the HBase Architecture

HBase storage architecture includes Client, HMaster, HRegionSever, HRegion, HLog, Store, MemStore, StoreFile and HFiles. The following figure shows HBase's storage architecture.

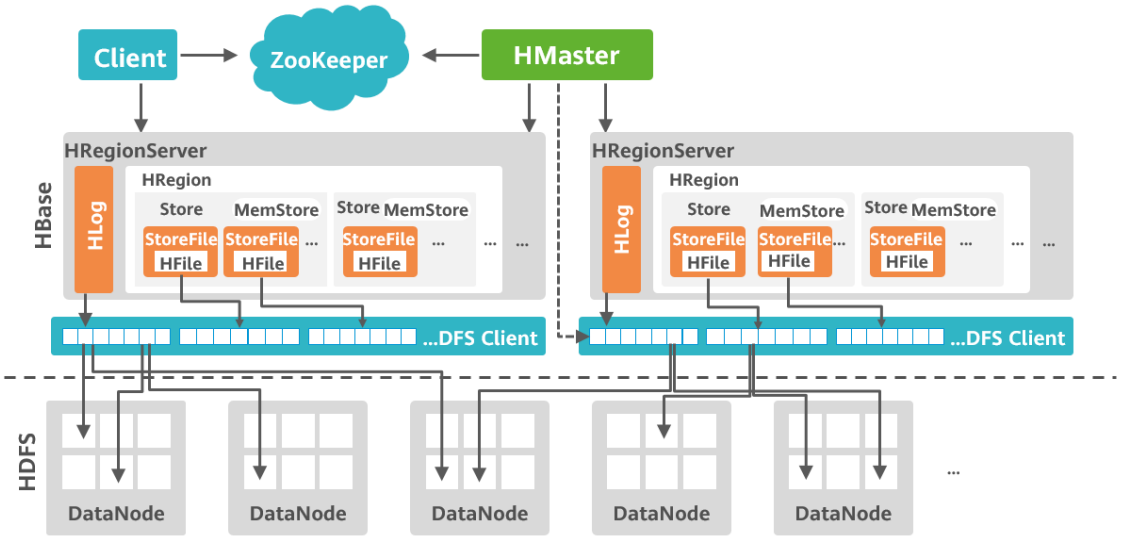


Figure 3-3 HBase storage architecture

Each table in HBase is divided into multiple sub-tables (HRegions) based on a certain range of keys. By default, if the size of an HRegion exceeds 256 MB, the HRegion is divided into two HRegions. This process is managed by HRegionServer, and the HRegion

allocation is managed by HMaster. As shown in the preceding figure, a client accesses data on HBase without the HMaster's participation. It accesses ZooKeeper and HRegionServer through addresses and accesses HRegionServer in data read/write mode. HMaster only maintains the metadata of tables and regions. The metadata of tables is stored on ZooKeeper, and the data load is low. When accessing an HRegion, HRegionServer creates an HRegion object, and then creates a Store object for each column family of the table. Each Store corresponds to a MemStore corresponding to zero or multiple StoreFiles. Each StoreFile corresponds to an HFile, which is the actual storage file. Therefore, the number of columns in an HRegion is the same as the number of Stores. One HRegionServer has multiple HRegions and one HLog.

HMaster has the following functions:

- Allocates HRegion to HRegionServer.
- Balances the load of HRegionServer.
- Detects and reallocates invalid HRegionServers.
- Recycles junk file in HDFS.
- Processes schema update requests.
- Manages users' operations on tables, such as adding, deleting, modifying, and querying tables.
- In HA mode, there is an active Master and a standby Master, preventing single point of failure (SPOF) of HMaster. Active Master: manages RegionServer in HBase, including the creation, deletion, modification, and query of a table, balances the load of RegionServer, adjusts the distribution of regions, splits regions and distributes regions after splitting, and migrates regions after RegionServer expires. Standby Master: takes over services when the active Master is faulty. The original active Master is demoted to the standby Master after the fault is rectified.

HRegionServer has the following functions:

- Maintains the HRegions allocated by HMaster and processes I/O requests to these HRegions.
- Divides the HRegion whose data size becomes excessively large during the running process.

HRegion: A table is divided into multiple HRegions horizontally. An HRegion is the minimum unit for distributed storage and load balancing in HBase. That is, different HRegions can be deployed on different HRegionServers. However, the same HRegion cannot be deployed on multiple HRegionServers. HRegion is divided by size. Generally, each table has only one HRegion. As data is continuously inserted into the table, the HRegion continues to increase. When a column cluster of the HRegion reaches a threshold (256 MB by default), the HRegion is divided into two new HRegions.

The HRegionServer to which the HRegions are allocated is completely dynamic. Therefore, a mechanism is required to locate the HRegionServer to which the HRegions are allocated. HBase uses a three-layer structure to locate the HRegions.

- Obtain the location of the **ROOT** table from the **/hbase/rs** file in ZooKeeper. The **ROOT** table has only one region.
- Search for the corresponding HRegion in the first table of the **META** table using the **ROOT** table. Actually, the **ROOT** table is the first region of the **META** table. Each Region in the **META** table has a record in the **ROOT** table.
- Use the **META** table to locate the HRegion of a user table. Each HRegion in the user table is a record in the **META** table. The **ROOT** table will never be divided into multiple HRegions, ensuring that any region can be located after a maximum of three hops. The client caches the queried location information and the cache will not proactively become invalid.

However, if all caches on the client become invalid, you need to perform the network roundtrip six times to locate the correct HRegion: three times to detect the cache failure, and another three times to obtain the HRegion location.

Store: Each HRegion consists of at least one Store. HBase stores the data to be accessed together in a Store. That is, HBase creates a Store for each column family. The number of Stores is the same as that of column families. A Store consists of a MemStore and zero or multiple StoreFiles. HBase determines whether to split an HRegion based on the Store size.

MemStore: is stored in the memory and stores the modified data, that is, keyValues. When the size of MemStore reaches a threshold (64 MB by default), MemStore is flushed to a file, that is, a snapshot is generated. Currently, HBase has a thread to perform the flush operation on MemStore.

StoreFile: After data in the MemStore memory is written to a file, the file becomes a StoreFile. The data at the bottom layer of the StoreFile is stored in the HFile format.

HFile: is a binary storage format for KeyValue data in HBase. The length of an HFile is not fixed, except for Trailer and FileInfo. In Trailer, pointers point to the start points of other data blocks. FileInfo records meta information about files. A data block is the basic unit of HBase I/O. To improve efficiency, HRegionServer provides the LRU-based block cache mechanism. The size of each data block can be specified by a parameter when a table is created. The default block size is 64 KB. Large blocks facilitate sequential scan, and small blocks facilitate random query. In addition to the Magic at the beginning of each data block, each data block is composed of KeyValue pairs. The Magic content is random numbers to prevent data damage.

HLog (WAL log): is the write ahead log, which is used for disaster recovery (DR). HLog records all data changes. Once an HRegionServer breaks down, data can be recovered from the logs.

LogFlusher: periodically writes information in the cache to log files. LogRoller: manages and maintains log files.

3.1.2.2 Data Read Process

When a user reads data, HRegionServer first accesses the MemStore cache. If the MemStore cache cannot be found, HRegionServer searches the StoreFile on the disk. The read process may be roughly divided into three steps:

- Sending a data read request from the client
- Locating the region

- Searching for data in the region

3.1.2.3 Data Write Process

The write process is similar to the read process and also contains three steps:

- Sending a data write request from the client
- Locating the region
- Writing data in the region

When a user needs to write data, the data is allocated to the corresponding HRegionServer for execution. User data is first written to HLog, then to MemStore, and finally to disks, generating a StoreFile. The **commit()** invocation returns the data to the client only after the operation is written to HLog.

3.1.2.4 Cache Update

The system periodically flushes data in the MemStore cache to the StoreFile file on the disk, clears the cache, and writes a flag to Hlog. A new StoreFile file is generated each time the data is flushed. Therefore, each Store contains multiple StoreFile files, and each HRegionServer has its own HLog file. Each time the HRegionServer is started, the HLog file is checked to determine whether a new write operation is performed after the latest cache flush. If an update is detected, the data is written to MemStore and then to StoreFile to provide services for users.

3.1.2.5 Merging StoreFiles

The required read latency prolongs as the number of HFiles increases. Therefore, a new StoreFile is generated each time data is written. However, if a large number of StoreFiles are generated, the search speed is affected. **Store.compact()** is called to combine multiple StoreFiles into one. The merge operation consumes a large number of resources. Therefore, only when the number of StoreFiles reaches a threshold, the merge operation is performed.

3.1.2.6 Working Principles of Store

Store is the core of HRegionServer. A Store consists of one MemStore and multiple StoreFiles. When the data volume of StoreFiles is too large, multiple StoreFiles are combined into one. When the size of a StoreFile is too large, a splitting operation is triggered, and one parent HRegion is split into two HRegions.

3.1.2.7 Working Principles of HLog

- In a distributed environment, system errors must be considered. HBase uses HLog to ensure system recovery.
- The HBase system configures an HLog file for each RegionServer, which is a write-ahead log (WAL).
- The updated data can be written to the MemStore cache only after the data is written to logs. In addition, the cached data can be written to the disk only after the logs corresponding to the data cached in the MemStore are written to the disk.
- ZooKeeper monitors the status of servers in each region in real time. When a server in a region is faulty, ZooKeeper notifies HMaster of the fault.

- HMaster first processes the remaining HLog files on the faulty RegionServer. The remaining HLog files contain log records from multiple regions.
- The system splits the HLog data based on the region to which each log belongs and saves the split data to the directory of the corresponding region. Then, the system allocates the invalid region to an available RegionServer and sends the HLog logs related to the region to the corresponding RegionServer.
- After obtaining the allocated region object and related HLogs, the RegionServer performs operations in the logs again, writes the data in the logs to the MemStore cache, and then updates the data to the StoreFile file on the disk for data restoration.
- Advantages of shared logs: The performance of writing data to tables is improved. Disadvantage: Logs need to be split during restoration.

3.1.3 HBase Performance Tuning

3.1.3.1 Impact of Multiple HFiles

The number of HFiles increases with the time because service data is continuously injected to the HBase cluster. The same query opens more files, so the query latency increases.

3.1.3.2 Compaction

HBase architecture is in Log-Structured Merge Tree mode. User data is written to WAL and then to the cache. When certain conditions are met, the cached data is flushed to disks and a data file HFile is generated. As more and more data is written, the number of flush times increases. As a result, the HFile data files increase. However, too many data files increase the I/O times due to data query. Therefore, HBase attempts to merge these files. This process is called compaction.

Some Hfile files from a Store of a region are selected for compaction. The compaction principle is simple. KeyValues are read from the data files to be merged, and then written to a new file after being sorted in ascending order. After that, the new file replaces all the files to be merged to provide services. Based on the compaction scale, HBase compactions are classified into minor compactions and major compactions.

Minor compaction is when small and adjacent StoreFiles are combined into a larger StoreFile. In this process, the cells that have been deleted or expired are not processed. A minor compaction results in fewer and larger StoreFiles.

Major compaction is when all StoreFiles are combined into one StoreFile. In this process, three types of meaningless data are deleted: deleted data, Time To Live (TTL) expired data, and data whose version number is higher than the specified one. In addition, major compaction takes a long time and consumes a large number of system resources, greatly affecting upper-layer services. Therefore, the function of automatically triggering major compaction is disabled for online services, and major compaction is manually triggered during off-peak hours.

3.1.3.3 OpenScanner

Before locating the corresponding RegionServer and Region of RowKeys, a Scanner needs to be opened to search for data. Because a Region contains MemStores and HFiles, a

Scanner needs to be opened for each of them to read data in MemStores and HFiles respectively. The scanner corresponding to HFile is StoreFileScanner. The scanner corresponding to MemStore is MemStoreScanner.

3.1.3.4 BloomFilter

Bloom Filter was proposed by Bloom in 1970. It is actually a long binary vector and a series of random mapping functions. Bloom Filter can be used to retrieve whether an element is in a set. Its advantage is that the space efficiency and query time are much longer than those of common algorithms. Its disadvantage is that it has a certain misidentification rate and makes data difficult to delete. In HBase, row keys are stored in HFiles. To query a row key from a series of HFiles, you can use Bloom Filter to quickly determine whether the row key is in the HFiles. In this way, most HFiles are filtered out, reducing the number of blocks to be scanned. Bloom Filter is critical to the random read performance of HBase. For the GET operation and some SCAN operations, HFiles that are not used can be deleted to reduce the actual I/O times and improve the random read performance.

3.1.3.5 Row Key

Row keys are stored in alphabetical order. Therefore, when designing row keys, you need to fully use the sorting feature to store together data that is frequently read and data that may be accessed recently. For example, if the data recently written to the HBase table is most likely to be accessed, you can use the timestamp as a part of the row key. Because the data is sorted in alphabetical order, you can use **Long.MAX_VALUE - timestamp** as the row key. In this way, newly written data can be quickly hit when being read.

3.1.3.6 Creating an HBase Secondary Index

In HBase, only row keys are used as the primary indexes. To retrieve and query data of non-row key fields in a database, distributed computing frameworks such as MapReduce or Spark are used. As a result, the hardware resource consumption and delay are high. To enable more efficient HBase data query and adapt to more scenarios, for example, response to non-rowkey field retrieval within seconds, or fuzzy query and multi-field combination query of each field, secondary indexes need to be created on HBase to meet more complex and diversified service requirements.

Currently, the famous open-source industrial solutions based on Coprocessor are: 1. H-Index: an HBase secondary index developed by Huawei based on Java and compatible with Apache HBase 0.94.8. It provides multiple table indexes, multiple column indexes, and indexes based on some column values. 2. Apache Phoenix: supports SQL on HBase and is compatible with multiple HBase versions. The secondary index is only one of the functions. The creation and management of secondary indexes are supported by the SQL syntax, which is easy to use. Currently, the community activeness and version update iteration of this project are in good condition.

3.1.4 Common HBase Shell Commands

create: creates a table.

list: lists all tables in HBase.

put: adds data to a specified cell in a table, row, or column. scan: browses information about a table.

get: obtains the value of a cell based on the table name, row, column, timestamp, time range, and version number.

drop: deletes a table.

3.2 Hive: Distributed Data Warehouse

3.2.1 Hive Overview and Architecture

3.2.1.1 Hive Overview

Hive is a data warehouse analysis system built based on Hadoop. It is used to extract, transform, and load data. It is a mechanism that can store, query, and analyze large-scale data stored in Hadoop. The Hive data warehouse tool can map structured data files to a database table, provide the SQL query function, and convert SQL statements into MapReduce tasks for execution. Hive has the advantage of low learning costs. It quickly collects MapReduce statistics by using SQL-like statements, simplifying the use of MapReduce without developing dedicated MapReduce applications.

Hive is suitable for statistical analysis of data warehouses. Hive supports most statements, such as data definition language (DDL) and data manipulation language (DML), as well as common aggregation functions, join queries, and conditional queries. It also provides methods for data extraction, transformation, and loading (ETL) to store, query, and analyze large-scale data sets stored in Hadoop, and supports User-Defined Function (UDF), User-Defined Aggregation Function (UDAF), and User-Defined Table Generating Function (UDTF). Additionally, **map** and **reduce** functions can also be customized, providing good scalability and scalability for data operations.

Hive is not suitable for online transaction processing and does not provide the real-time query function. It is best suited for batch job processing based on a large amount of immutable data. Hive features scalability (dynamically adding devices to the Hadoop cluster), scalability, fault tolerance, and loose coupling of the input format.

3.2.1.2 Comparison Between Hive and Traditional Data Warehouses

Before Hive became popular, most enterprises used the traditional parallel data warehouse architecture. Traditional data warehouses usually use large-scale servers and mature solutions from well-known vendors outside China. They are expensive and have inflexible scalability. In addition, platform tools are difficult to adapt to other vendors, user experience is poor, and data development is inefficient. When the data volume reaches the terabytes (TB) level, performance deteriorates. Furthermore, traditional data warehouses are only good at processing only structured or semi-structured data and cannot well support the processing of unstructured data. Hive provides enterprises with cost-effective and high-quality solutions. The following table shows the comparison between Hive and traditional data warehouses.

Table 3-3 Comparison between Hive and traditional data warehouses

Feature	Hive	Traditional Data Warehouse
Storage	HDFS is used to store data. Theoretically, infinite expansion is possible.	Clusters are used to store data, which have a capacity upper limit. With the increase of capacity, the computing speed decreases sharply. Therefore, data warehouses are applicable only to commercial applications with small data volumes.
Executor	The default execution engine is Tez.	More efficient algorithms can be selected to perform queries, or more optimization measures can be taken to speed up the queries.
Usage	HQL (SQL-like)	SQL
Flexibility	Metadata storage is independent of data storage, decoupling metadata and data.	Low flexibility. Data can be used for limited purposes.
Analysis speed	Computing depends on the cluster scale and the cluster expands easily. In the case of a large amount of data, computing is much faster than that of a common data warehouse.	When the data volume is small, the data processing speed is high. When the data volume is large, the speed decreases sharply.
Index	Low efficiency	Productive
Ease of use	Application models need to be self-developed. The flexibility is high, but the usability is low.	A set of mature report solutions is integrated to facilitate data analysis.
Reliability	Data is stored in HDFS, implementing high data reliability and fault tolerance.	Reliability is low. If a query fails, the task needs to be started again. The data fault tolerance depends on hardware RAID.

Feature	Hive	Traditional Data Warehouse
Environment dependency	Low dependency on hardware, applicable to common machines	High dependency on high-performance business servers
Price	Open-source product	Expensive in commercial use

3.2.2 Hive Functions and Architecture

3.2.2.1 Hive Architecture

Main components of Hive include UI components, Driver components (Compiler, Optimizer, and Executor), Metastore components, JDBC/ODBC, Thrift Server, and Hive Web Interface (HWI). The following describes these components.

- Driver is the core component of Hive. It consists of Compiler, Optimizer, and Executor. These components are used to parse Hive SQL statements, compile and optimize the statements, generate execution plans, and invoke the underlying MapReduce computing framework.
- MetaStore is used by Hive to manage metadata. Hive metadata is stored in relational databases. The supported relational databases are Derby and MySQL. Derby is the default database used by Hive and is embedded in Hive. However, the database supports only a single session and is not applicable to the production where multiple sessions are generated. Therefore, MySQL is used as the metadata database.
- Thrift Server provides JDBC and ODBC access capabilities for developing an extensible cross-language service. Hive integrates this service to enable different programming languages to invoke Hive APIs.
- Web Interface is provided by the Hive client to access Hive through web pages.

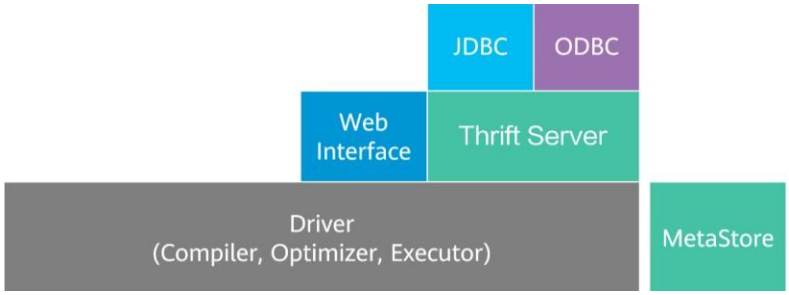


Figure 3-4 Hive architecture

3.2.2.2 Hive Data Storage Model

Hive data is classified into table data and metadata. Table data is the data in Hive tables. Metadata stores table names, table columns and partitions and their attributes, table attributes (whether the table is an external table), and directories where table data is stored. The following describes internal and external tables:

- Internal table: An internal table in Hive is similar to a table in a relational database. Each table has a directory in HDFS for storing table data. The directory can be configured using the **hive.metastore.warehouse.dir** attribute in the **${HIVE_HOME}/conf/hive-site.xml** configuration file, the default value of this attribute is **/user/hive/warehouse** (the directory is in HDFS). You can change the value based on the site requirements.
- External table: An external table is similar to an internal table, but its data is not stored in the directory to which the external table belongs. Instead, the data is stored in another directory. In this way, if you delete the external table, the data pointed by the external table will not be deleted. Only the metadata corresponding to the external table is deleted. If you delete a table, all data in the table, including metadata, is deleted.
- Partition: Data in a table is divided into multiple subdirectories for storage. A table can have one or more partitions. A partition corresponds to a directory in the table, and data of all partitions is stored in the corresponding directory. The standard process for Hive partitioning is to specify a partition column during table creation. Note that the partition column is not a field in the table, but an independent column, and the partition column can be specified as a query condition.
- Bucketing: The hash value is calculated based on the specified column, and data is divided based on the calculated hash value. Then, the remainder obtained by dividing the hash value by the number of buckets is used to determine the bucket where the data is stored. Each bucket is a file. The number of buckets is specified when a table is created, and the buckets can be sorted. Bucketing is mainly used to sample data and improve the efficiency of query operations.

As shown in the following table, a database stores tables where partitions, buckets, skewed data, and normal data are involved. Buckets can also be created in a partition.

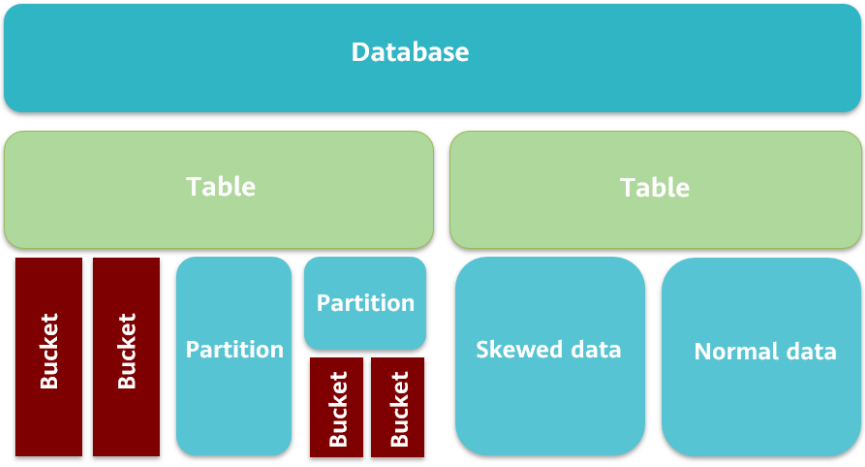


Figure 3-5 Data storage model of Hive

3.2.3 Hive Basic Operations

3.2.3.1 Hive SQL

Data definition language (DDL) is used to create, delete, modify tables, partitions, and data types.

Create a database:

```
CREATEDATABASE|SCHEMA[IFNOTEXISTS]<databasename>
```

Delete the database:

```
DROP(DATABASE|SCHEMA)[IFEXISTS]database_name[RESTRICT|CASCADE]
```

Data manipulation language (DML) is used to import and export data.

Data import:

```
LOADDATA[LOCAL]INPATH'filepath' [OVERWRITE]INTOTABLE
tablename[PARTITION(partcol1=val1,partcol2=val2...)]
```

Data query language (DQL) is used to perform simple queries, complex queries, GROUP BY, ORDER BY, JOIN, and more.

Basic SELECT statement:

```
SELECT[ALL|DISTINCT]select_expr,select_expr,...FROMtable_reference
[WHEREwhere_condition]
[GROUPBYcol_list[HAVINGcondition]][CLUSTERBYcol_list
|[DISTRIBUTEBYcol_list][SORTBY|ORDERBYcol_list]
]
[LIMITnumber]
```

3.3 Quiz

1. What are the similarities and differences between Hive and RDBMS?
2. Why is it not recommended to use too many column families in HBase?

4 ClickHouse — Online Analytical Processing Database Management System

4.1 Overview

4.1.1 Introduction

ClickHouse is an OLAP colum-oriented database management system. It is independent of the Hadoop big data system and features ultimate compression rate and fast query. It supports SQL query and delivers superior query performance, especially the aggregation analysis and query performance based on large and wide tables. Its query speed is one order of magnitude faster than that of other analytical databases.

ClickHouse is an OLAP colum-oriented database management system. It is independent of the Hadoop big data system and features ultimate compression rate and fast query. It supports SQL query and delivers superior query performance, especially the aggregation analysis and query performance based on large and wide tables. Its query speed is one order of magnitude faster than that of other analytical databases.

4.1.2 Advantages

The core functions of ClickHouse are as follows:

- Comprehensive DBMS functions

  ClickHouse has comprehensive database management functions, including the basic functions of a DBMS:

  Data Definition Language (DDL): allows databases, tables, and views to be dynamically created, modified, or deleted without restarting services.

  Data Manipulation Language (DML): allows data to be queried, inserted, modified, or deleted dynamically.

  Permission control: supports user-based database or table operation permission settings to ensure data security.

  Data backup and restoration: supports data backup, export, import, and restoration, meeting the requirements of the production environment.

  Distributed management: provides the cluster mode to automatically manage multiple database nodes.

- Column-oriented storage and data compression

ClickHouse is a database that uses column-based storage. Data is organized by column. Data in the same column is stored together, and data in different columns is stored in different files.

During data query, column-based storage can reduce the data scanning range and data transmission size, thereby improving data query efficiency.

- Vectorized execution engine

  ClickHouse uses CPU's Single Instruction Multiple Data (SIMD) for vectorized execution. SIMD is an implementation mode that uses a single instruction to operate multiple pieces of data and improves performance with data parallelism (other methods include instruction-level parallelism and thread-level parallelism). The principle of SIMD is to implement parallel data operations at the CPU register level.

- Relational model and SQL query

  ClickHouse uses SQL as the query language and provides standard SQL query APIs for existing third-party analysis visualization systems to easily integrate with ClickHouse.

  In addition, ClickHouse uses relational models so the cost of migrating systems built on traditional relational databases or data warehouses to ClickHouse is lower.

- Data sharding and distributed query

  The ClickHouse cluster consists of one or more shards, and each shard corresponds to one ClickHouse service node. The maximum number of shards depends on the number of nodes (one shard corresponds to only one service node).

  ClickHouse introduces the concepts of local table and distributed table. A local table is equivalent to a data shard. A distributed table itself does not store any data. It is an access proxy of the local table and functions as the sharding middleware. With distributed tables, multiple data shards can be accessed using the proxy, thereby implementing distributed query.

- Excellent performance

  Excellent query and write performance

4.1.3 Use Cases

ClickHouse is a combination of Clickstream and Data Warehouse. It is initially applied to a web traffic analysis tool to perform OLAP analysis for data warehouses based on page click event flows. Currently, ClickHouse is widely used in Internet advertising, app and web traffic analysis, telecommunications, finance, and Internet of Things (IoT) fields. It is applicable to business intelligence application scenarios and has a large number of applications and practices worldwide. For details, visit **https://clickhouse.tech/docs/en/introduction/adopters/**.

4.2 Architecture and Basic Features

4.2.1 Architecture

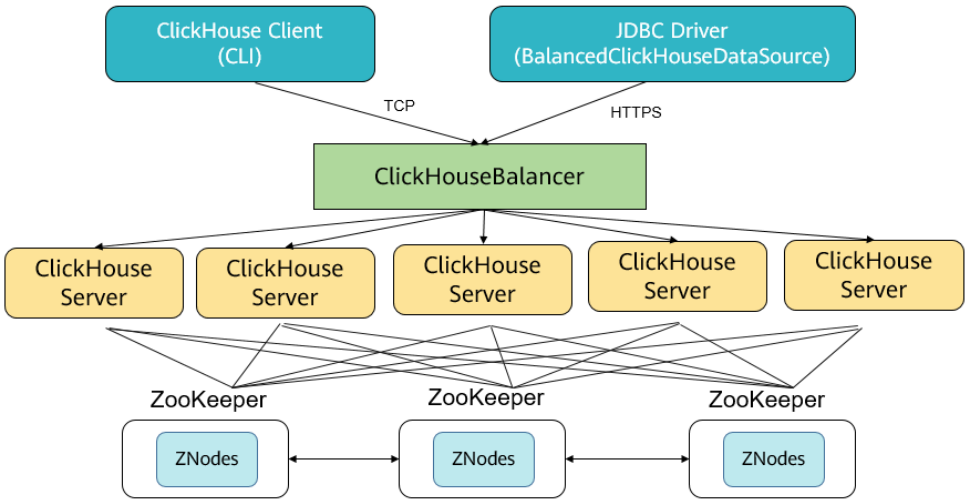


Figure 4-1 Architecture

ClickHouse can be accessed by CLI and JDBC clients using either TCP or HTTP.

ClickHouseBalancer can balance the load of multiple ClickHouseServers to improve the reliability of application access.

ClickHouseServer provides open-source server capabilities. It implements the multi-master replica mechanism using ZooKeeper and the ReplicatedMergeTree engine.

4.2.2 Data Shards vs. Copies

From the perspective of data, assume that *N* nodes form a ClickHouse cluster, and each node has a table Y with the same structure. If data in table Y of N1 is completely different from the data in table Y of N2, N1 and N2 are shards of each other. If their data is the same, they are copies of each other. In other words, data between shards is different, but data between copies is exactly the same.

From the perspective of functions, the main purpose of using copies is to prevent data loss and increase data storage redundancy, while the main purpose of using shards is to implement horizontal data partitioning.

4.2.3 Characteristics of ClickHouse Copies

Table-level copies depend on ZooKeeper's distributed collaboration.

Copies use the multi-host architecture. Executing **INSERT** and **ALTER** statements on any copy has the same effect.

Blocks: Data that is written is divided into multiple blocks based on **max_insert_block_size**.

Atomicity: All blocks are successfully written or fail to write.

Uniqueness: Hash message digests are calculated based on indicators such as data sequence, row, and size to prevent blocks from being repeatedly written due to exceptions.

Copies increase data storage redundancy and reduce data loss risks. The multi-host architecture enables each copy to act as the entry for data read and write, sharing the load of nodes.

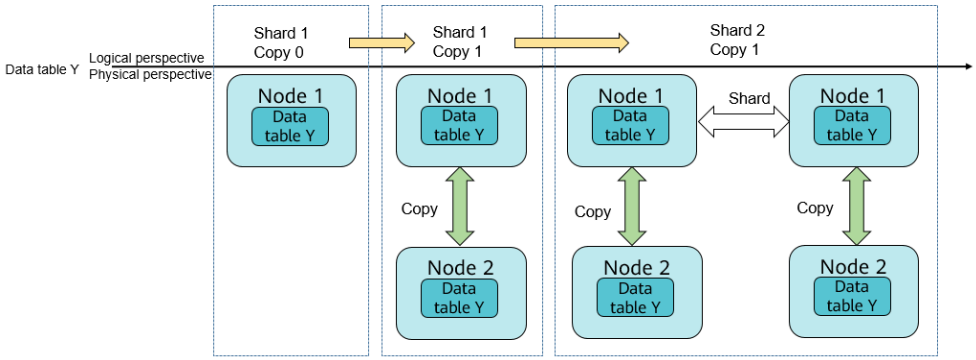


Figure 4-2 ClickHouse copies

4.3 Enhanced Features

4.3.1 Online Scale-Out and Data Migration

As the service volume increases, the cluster storage capacity or CPU resources are not enough. MRS provides the ClickHouse data migration tool to migrate some partitions of one or more partitioned MergeTree tables on several ClickHouseServer nodes to the same tables on other ClickHouseServer nodes. This ensures service availability while implementing smooth scale-out.

When adding ClickHouse nodes to a cluster, you can use this tool to migrate some data from the original nodes to the new nodes to balance data after scale-out.

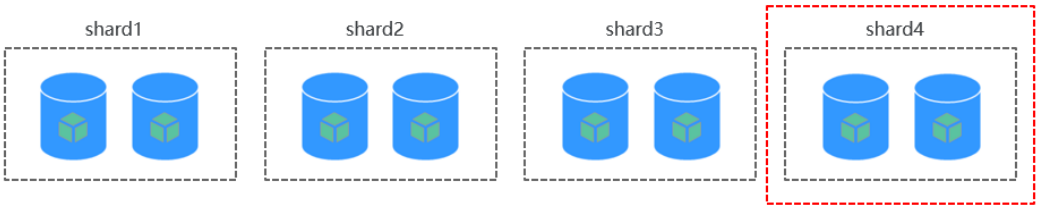


Figure 4-3 Online scale-out and data migration

4.3.2 Rolling Upgrade/Restart

As shown in Figure 4-3, there is no central node in the ClickHouse cluster. The cluster is more of a static resource pool. To use the ClickHouse cluster, you need to pre-define the cluster information in the configuration file of each node. Only in this way, services can be correctly accessed.

Figure 4-4 ClickHouse cluster

In general database systems, details such as data partitions and copy storage below the table level are typically hidden, and users are unaware of them. But ClickHouse requires you proactively plan and define detailed configurations such as shards, partitions, and copy storage locations. This brings poor user experience. To eliminate this inconvenience, MRS ClickHouse instances pack the manual work for centralized management, making MRS ClickHouse much easier to use. A ClickHouse instance consists of three ZooKeeper nodes and multiple ClickHouse nodes. The Dedicated Replica mode is used to ensure HA using two copies.

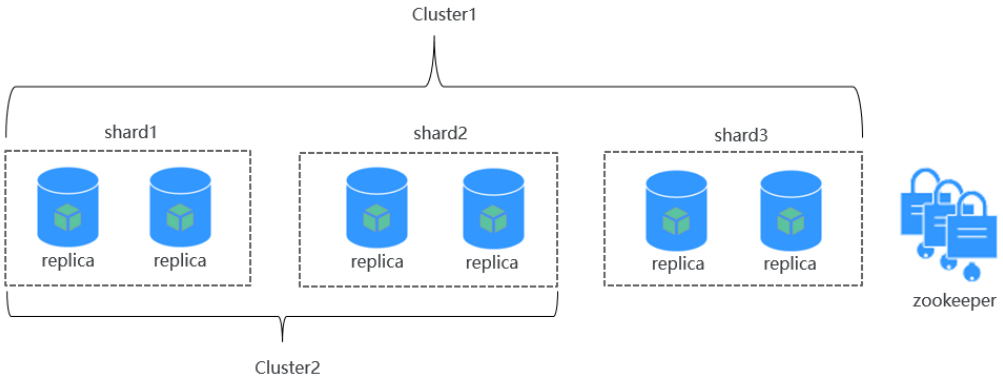


Figure 4-5 ClickHouse cluster structure

4.3.3 Automatic Cluster Topology and ZooKeeper Overload Prevention

MRS uses the ELB-based high availability (HA) deployment architecture to automatically distribute user access traffic to multiple backend nodes, expanding service capabilities to external systems and improving fault tolerance. As shown in Figure 4-5, when a client application requests a cluster, Elastic Load Balance (ELB) is used to distribute traffic. With ELB polling, data is written to local tables and read from distributed tables on different nodes. In this way, data read/write load and high availability of application access are guaranteed.

After a ClickHouse cluster is provisioned, each ClickHouse node corresponds to one copy. Two copies form a logical shard. For example, when creating a ReplicatedMergeTree engine table, you can specify a shard. In this way, data of two copies in the same shard can be automatically synchronized.

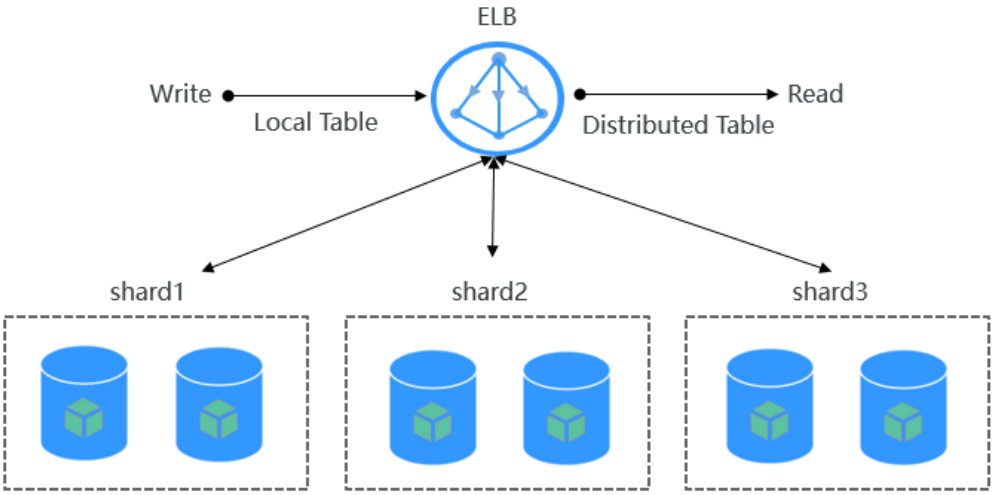


Figure 4-6 HA deployment architecture

4.3.4 Peripheral Ecosystem Interconnection

General report data (detail flat-wide tables) is generated by Flink stream computing applications and written to ClickHouse in quasi-real time. General report data (detail flat-wide tables) is generated by Hive or Spark jobs and imported to ClickHouse in batches.

4.4 Quiz

1. What table engines does ClickHouse offer?

5 MapReduce and YARN Technical Principles

5.1 Overview

5.1.1 Introduction to MapReduce

Hadoop has experienced three major versions since its release: 1.0, 2.0, and 3.0. The most typical versions are 1.0 and 2.0. Compared with 2.0, 3.0 does not change much. The architecture of Hadoop 1.0 is simple, including only HDFS and MapReduce. The lower layer is the open source Hadoop distributed file system (HDFS) implemented by Google File System (GFS), while the upper layer is the distributed computing framework MapReduce. MapReduce of Hadoop 1.0 does not separate resource management from job scheduling. As a result, when multiple jobs are submitted concurrently, the resource scheduler is overloaded, resulting in low resource utilization. In addition, Hadoop 1.0 does not support heterogeneous computing frameworks. In Hadoop 2.0, YARN, the resource management and scheduling system, is introduced to replace the original computing framework, implementing proper resource scheduling. YARN is compatible with multiple computing frameworks.

MapReduce is designed and developed based on the MapReduce paper released by Google. MapReduce is used for parallel computing and offline computing of large-scale data sets (larger than 1 TB). MapReduce can be understood as a process of summarizing a large amount of disordered data based on certain features (Map) and processing the data to obtain the final result (Reduce).

A major advantage of MapReduce is that it has a highly abstract programming idea. To develop a distributed program, developers only need to use some simple APIs without considering other details, such as data shards and data transmission. Developers only need to focus on the program's logic implementation. Another advantage is scalability. When the data volume reaches a certain level, the existing cluster cannot meet the computing and storage requirements. In this case, you can add nodes to scale out a cluster.

MapReduce also features high fault tolerance. In a distributed environment, especially as the cluster scale increases, the failure rate of the cluster also increases, which may cause task failures and data loss. Hadoop uses computing or data migration policies to improve cluster availability and fault tolerance.

However, MapReduce also has its own limitations. (1) MapReduce can only process tasks that can be divided into multiple independent subtasks. (2) All data on which MapReduce depends comes from files. The entire computing process involves a large number of file

I/Os, which inevitably affects the computing speed. So, it is not friendly to process real-time data.

5.1.2 Introduction to YARN

Apache Hadoop Yet Another Resource Negotiator (YARN) is a resource management system. It provides unified resource management and scheduling for upper-layer applications, remarkably improving cluster resource utilization, unified resource management, and data sharing.

In Hadoop 1.0, resources are scheduled using MRv1. This version has the following disadvantages:

- The master node has SPOFs. The fault recovery depends on the periodic checkpoint, which does not ensure the reliability. When a fault occurs, users are only notified of the fault and they determine whether to perform recalculation.
- Job scheduling and resource scheduling are not distinguished. When MapReduce is running, a large number of concurrent jobs exist in the environment. Therefore, diversified and efficient scheduling policies are important.
- When a large number of jobs are concurrently executed without considering resource isolation and security, the major issues are how to ensure that a single job does not occupy too many resources and that users' programs are secure for the system.

With the introduction of YARN in Hadoop 2.0, these disadvantages are eliminated. YARN is a lightweight elastic computing platform that supports Spark and Storm frameworks as well as MapReduce.

5.2 Functions and Architectures of MapReduce and YARN

5.2.1 MapReduce Process

As the name implies, the MapReduce calculation process can be divided into two phases: Map and Reduce. The output in the Map phase is the input in the Reduce phase.

MapReduce can be understood as a process of summarizing a large amount of disordered data based on certain features and processing the data to obtain the final result. In the Map phase, keys and values (features) are extracted from each piece of uncorrelated data. In the Reduce phase, data is organized by keys followed by several values. These values are correlated. On this basis, further processing may be performed to obtain a result. The MapReduce working process is as follows:

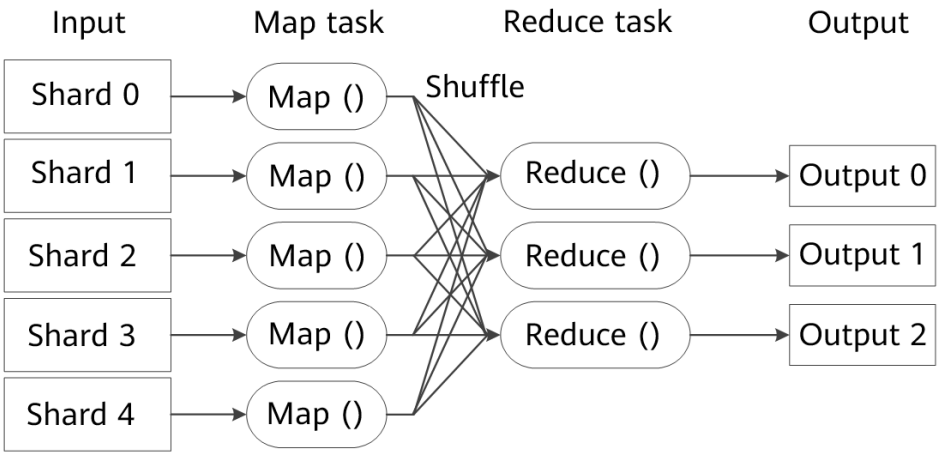


Figure 5-1 MapReduce working process

In the preceding figure, different Map tasks do not communicate with each other, and no information is exchanged between different Reduce tasks. Users cannot explicitly send messages from one machine to another. All data exchange is implemented through the MapReduce framework.

5.2.1.1 Map Phase

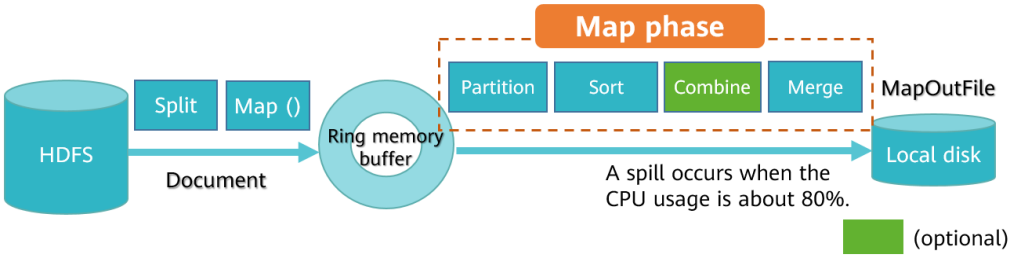


Figure 5-2 Map workflow

Before jobs are submitted, the files to be processed are split. By default, the MapReduce framework regards a block as a split. Client applications can redefine the mapping between blocks and splits.

In the Map phase, data is stored in a ring memory buffer. When the data in the buffer reaches about 80%, spill occurs. In this case, the data in the buffer needs to be written to the local disk. The following operations need to be performed before data is written to the local disk:

- Partition: By default, the hash algorithm is used for partitioning. The MapReduce framework determines the number of partitions based on that of Reduce tasks. The records with the same key value are sent to the same Reduce tasks for processing.
- Sort: The outputs of Map are sorted, for example, ('Hi','1'), ('Hello','1') is reordered as ('Hello','1'),('Hi','1').
- Combine: This action is optional in the MapReduce framework by default. For example, you can combine ('Hi','1'), ('Hi','1'), ('Hello','1'), and (Hello','1') into ('Hi','2'). ('Hello','2').
- Merge: After a Map task is processed, many spill files are generated. These spill files must be merged to generate a partitioned and sorted spill file MapOutFile (MOF). To

reduce the amount of data to be written to disks, MapReduce allows MOFs to be written after being compressed.

5.2.1.2 Reduce Phase

MOF files need to be sorted. If the amount of data received by Reduce tasks is small, the data is directly stored in the buffer. As the number of files in the buffer increases, the MapReduce background thread merges the files into a large file. Many intermediate files are generated during the merge operation. The last merge result is directly outputted to the Reduce function defined by the user. When the data volume is small, the data does not need to be spilled to the disk. Instead, the data is merged in the cache and then output to Reduce. Typically, when the MOF output progress of a Map task reaches 3%, Reduce task is started to obtain MOF files from each Map task. The number of Reduce tasks is determined by clients and determines the number of MOF partitions. For this reason, the MOF files outputted by Map tasks map to Reduce tasks. The process is as follows:

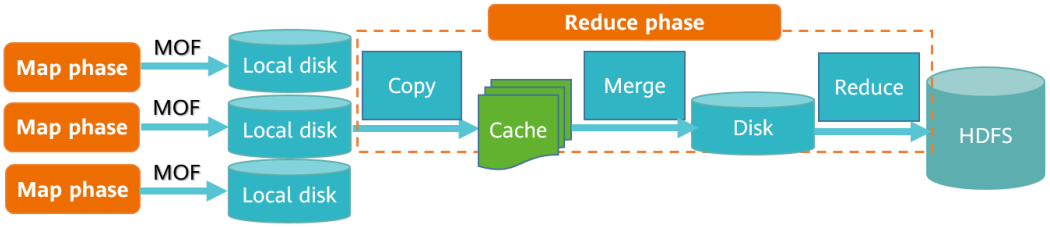


Figure 5-3 Reduce workflow

5.2.1.3 Shuffle Process

Shuffle is a process of transferring intermediate data between the Map and Reduce phases, including obtaining MOF files by Reduce tasks from Map tasks and sorting and merging MOF files.

Before all Map tasks are completed, the system merges the files into a large file and saves the file on the local disk. During file merging, if the number of spilt files is greater than the preset value (3 by default), the combiner can be started again. JobTracker keeps monitoring the execution of Map tasks and notifies Reduce tasks to obtain data. The Reduce task queries whether the Map task is completed from the JobTracker through the remote procedure call (RPC). If the Map task is completed, the Reduce task obtains data. The obtained data is stored to the cache, the data is merged from different Map machines, and then written to disks. Multiple spill files are merged into one or more large files, and key-value pairs in the files are sorted.

A typical wordcount program is as follows:

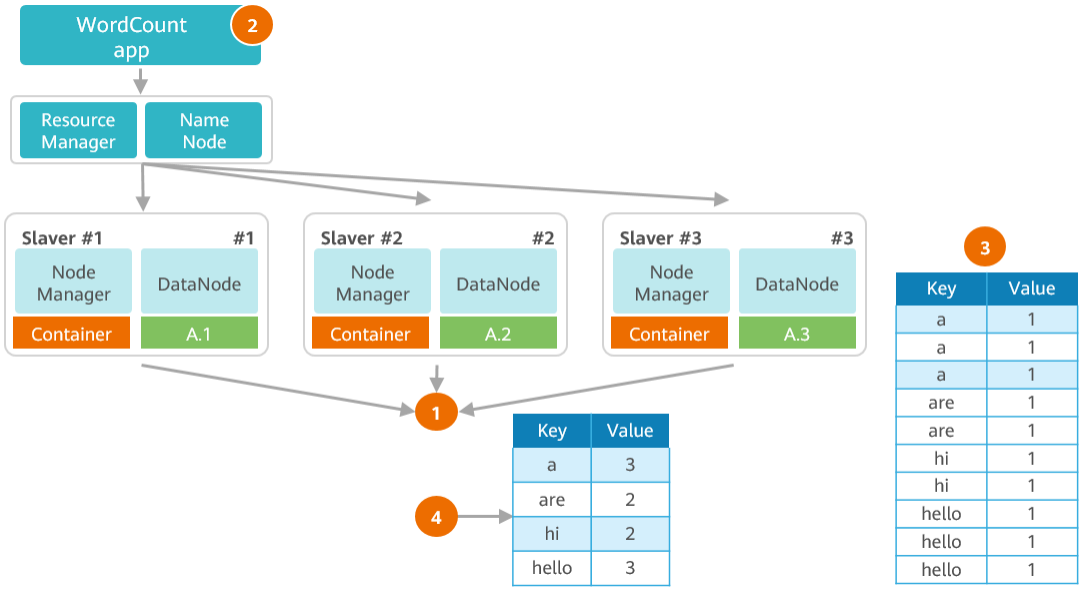


Figure 5-4 Example WordCount program

If a user needs to analyze the frequency of each word in text file A. The MapReduce framework can quickly hep with the statistical analysis.

Step 1 File A is stored on HDFS and divided into blocks A.1, A.2, and A.3 that are stored on DataNodes #1, #2, and #3.

Step 2 The WordCount analysis and processing program provides user-defined Map and Reduce functions. WordCount submits analysis applications to ResourceManager. Then ResourceManager creates jobs based on the request and create three Map tasks as well as three Reduce tasks that are running in a container.

Step 3 Map tasks 1, 2, and 3 output an MOF file that is partitioned and sorted but not combined. For details, see the table.

Step 4 Reduce tasks obtain the MOF file from Map tasks. After combination, sorting, and user-defined Reduce logic processing, statistics shown in the table is output.

5.2.1.4 YARN Architecture

Figure 5-6 shows the YARN architecture, including the ResourceManager, ApplicationMaster, NodeMananger, Container, and Client modules.

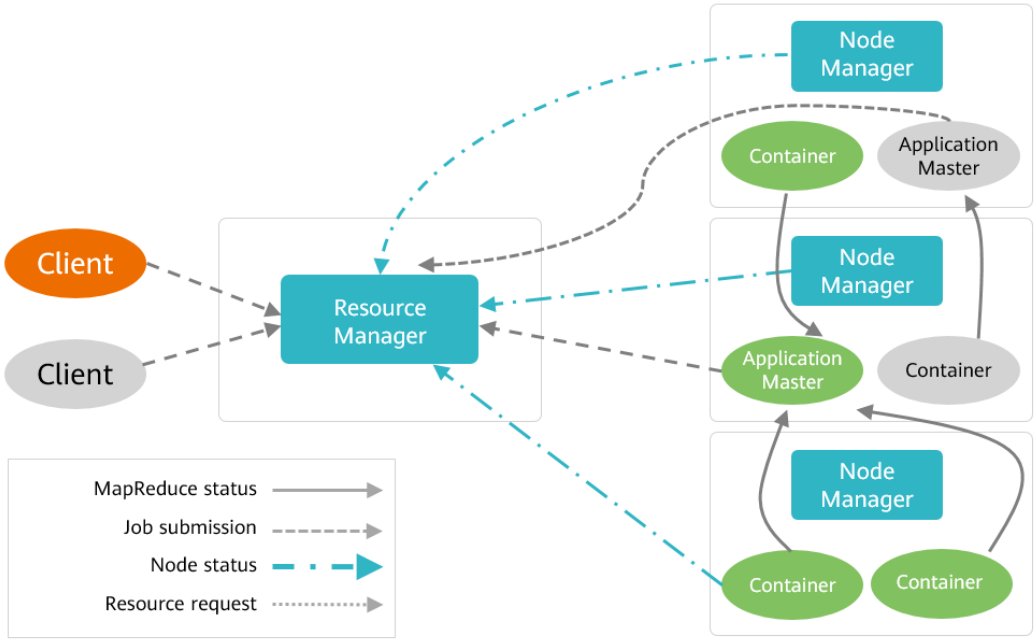


Figure 5-5 YARN architecture

There is only one ResourceManager available in a cluster for centralized resource management, scheduling, and allocation. NodeManager is the agent of a single node in YARN. Each node has one NodeManager as the agent to monitor the resource usage (CPU, memory, disk, and network) of the node and report the node status to ResourceManager.

ApplicationMaster executes and schedules data processing jobs. ApplicationMaster communicates with ResourceManager to obtain resources for computing. After obtaining resources, ApplicationMaster communicates with NodeManager on each node, summarizes tasks in the allocated containers, and monitors the task execution status. (Each time a Client submits an application, an ApplicationMaster is created.) ApplicationMaster applies for container resources from ResourceManager. After obtaining the resources, ApplicationMaster sends the application to run to the container for startup, and then performs distributed computing.

A container is an abstraction of resources in YARN. It encapsulates a certain number of resources (such as CPU and memory) on a node. When ApplicationMaster applies for resources from ResourceManager, ResourceManager returns containers to ApplicationMaster.

5.2.1.5 Job Scheduling Process of MapReduce on YARN

Figure 5-7 shows the task scheduling process of MapReduce on YARN.

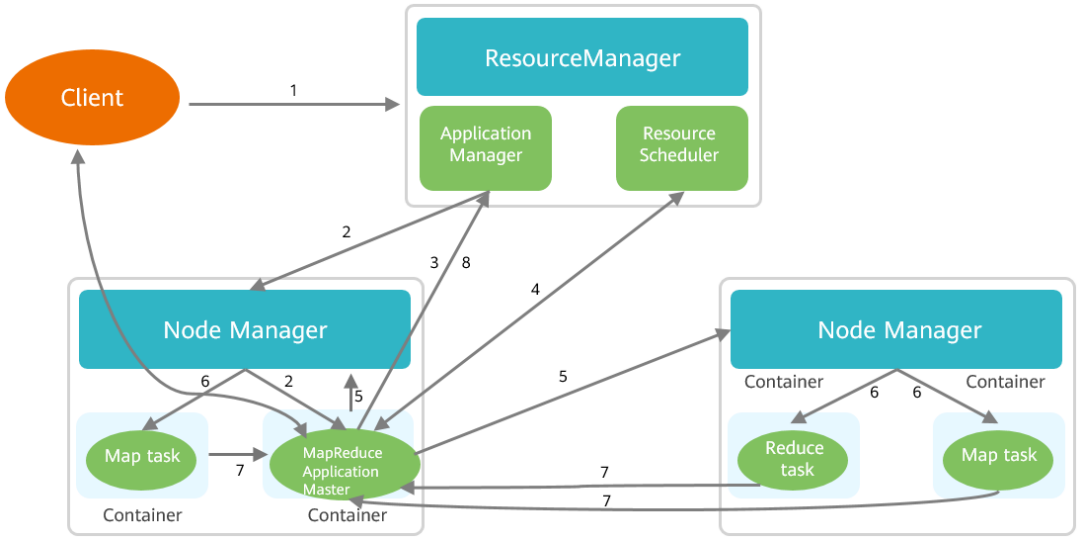


Figure 5-6 Job scheduling process of MapReduce on YARN

Step 1 A client submits applications to YARN, including ApplicationMaster, commands for starting ApplicationMaster, and user programs.

Step 2 ResourceManager allocates the first container to the applications and asks NodeManager to start ApplicationMaster in the container.

Step 3 ApplicationMaster registers with ResourceManager so users can directly view the operating status of the applications. Then ResourceManager applies for resources for each task and monitors the tasks until the task running ends (that is, repeat steps 4 to 7).

Step 4 ApplicationMaster applies for and obtains resources from ResourceManager in polling mode using the RPC protocol.

Step 5 Once obtaining resources, ApplicationMaster asks NodeManager to start the tasks.

Step 6 After setting an operating environment (including environment variables, JAR file, and binary programs) for the tasks, NodeManager writes task startup commands into a script and runs this script to start the tasks.

Step 7 Each task uses RPC to report status and progress to ApplicationMaster so that ApplicationMaster can restart a task if the task fails. During application running, you can use RPC to obtain their statuses from ApplicationMaster at any time.

Step 8 After the applications end, ApplicationMaster deregisters itself from ResourceManager and closes itself.

5.2.1.6 Fault Tolerance Mechanism of YARN ApplicationMaster

The YARN architecture provides a sound fault tolerance mechanism to handle ResourceManager and NodeManager running failures with different solutions.

- ResourceManager high availability solution

ResourceManager in YARN is responsible for resource management and task scheduling of the entire cluster. The YARN HA solution uses redundant ResourceManager nodes to solve SPOFs.

ResourceManager HA is achieved using active-standby ResourceManager nodes. Similar to the HDFS HA solution, the ResourceManager HA allows only one ResourceManager node to be in the active state at any time. When the active ResourceManager fails, an active-standby switchover can be triggered automatically or manually.

When automatic failover is not enabled, after the YARN cluster is enabled, administrators need to run the **yarnrmadmin** command to manually switch one of the ResourceManager nodes to the active state. Upon a planned maintenance event or a fault, they are expected to first demote the active ResourceManager to the standby state and the standby ResourceManager promote to the active state.

When automatic switchover is enabled, a built-in ZooKeeper-based ActiveStandbyElector determines which ResourceManager node should be the active one. When the active ResourceManager is faulty, another ResourceManager node is automatically elected to be the active one to take over the faulty node.

- YarnAppMaster fault tolerance mechanism

  When ResourceManager nodes in the cluster are deployed in HA mode, the addresses of all these nodes need to be added to the **yarn-site.xml** configuration file used by clients. Clients (including ApplicationMaster and NodeManager) try connecting to the ResourceManager nodes in polling mode until they hit the active ResourceManager node. If the active ResourceManager cannot be connected with, the client continuously searches for a new one in polling mode.

  In YARN, ApplicationMasters (AMs) run on NodeManagers, similar to containers.

  (Unmanaged AMs are ignored.) ApplicationMasters may break down, exit, or shut down. If an ApplicationMaster node goes down, ResourceManager kills all the containers of that ApplicationAttempt, including all containers running on NodeManagers. ResourceManager starts a new ApplicationAttempt on another compute node.

  Different types of applications need to handle ApplicationMaster restart events in multiple ways. The objective of MapReduce applications is to prevent task status loss but allow some status loss. However, for a long-period service, users do not want the entire service to stop due to an ApplicationMaster fault.

  YARN can retain the status of the container when a new ApplicationAttempt is started so running jobs can continue to run without faults.

5.3 Resource Management and Task Scheduling of YARN

5.3.1 YARN Resource Management

In YARN, the number of memory and CPUs that can be allocated to each NodeManager node can be set by configuring the following three parameters:

- **yarn.nodemanager.resource.memory-mb** indicates the size of the physical memory that can be allocated to the container on NodeManager, in MB. The value must be smaller than the memory size of the NodeManager server.
- **yarn.nodemanager.vmem-pmem-ratio** indicates the ratio of the maximum available virtual memory to the physical memory of a container. The assigned container memory usage is specified based on physical memory. The difference between the virtual memory usage and assigned container memory usage is less than the parameter value.
- **yarn.nodemanager.resource.cpu-vcore** indicates the number of vCores that can be allocated to a container. It is recommended that this parameter value be set to 1.5 to 2 times the number of vCores.

In Hadoop 3.*x*, the YARN resource model has been promoted to support user-defined countable resource types, not just CPU and memory. Common countable resource types include GPU resources, software licenses, and locally-attached storage in addition to CPU and memory, but do not include ports and labels.

5.3.2 YARN Resource Schedulers

Ideally, the requests for Yarn resources should be met immediately. However, in reality, resources are limited. Especially in a busy cluster, it usually takes a period of time for an application resource request to obtain the corresponding resources. In YARN, a scheduler allocates resources to applications. There are three schedulers available based on different policies:

- FIFO scheduler: Applications are arranged into a queue based on the submission sequence. This queue is a first-in-first-out (FIFO) queue. During resource allocation, resources are first allocated to the application on the top of the queue. After the application on the top of the queue meets the requirements, resources are allocated to the next application. The rest may be deduced by analogy.
- Capacity scheduler: Multiple organizations are allowed to share the entire cluster. Each organization can obtain some computing capabilities of the cluster. A dedicated queue is allocated to each organization, and certain cluster resources are allocated to each queue. Multiple queues are configured to provide services for multiple organizations. In addition, resources in a queue can be divided so that multiple members in an organization can share the queue resources. In a queue, the FIFO policy is used for resource scheduling.
- Fairs scheduler: All applications have an equal share of resources. (The fairness can be configured.)

5.3.3 Introduction to Capacity Scheduler

Capacity scheduler enables Hadoop applications to run in a shared, multi-tenant cluster while maximizing the throughput and utilization of the cluster. It allocates resources by queue. Upper and lower limits are specified for the resource usage of each queue. Each user can set the upper limit of resources that can be used. Administrators can restrict the resource used by a queue, user, or job. Job priorities can be set but resource preemption is not supported. In Hadoop 3.*x*, OrgQueue expands the capacity scheduler and provides a programmatic way to change the queue configuration through the REST API. In this

way, administrators can automatically configure and manage the queue in the **administer_queue ACL** of the queue.

The capacity scheduler has the following features:

- **Capacity assurance**: Administrators can set upper and lower limits for the resource usage of each queue. All applications submitted to the queue share the resources.
- **Flexibility**: The remaining resources of a queue can be used by other queues that require resources. If a new application is submitted to the queue, the resources released by other queues will be returned to the queue.
- **Priority scheduling**: Queues support task priority scheduling (FIFO by default).
- **Multi-leasing**: A cluster can be shared by multiple users or applications. Administrators can add multiple restrictions to prevent cluster resources from being exclusively occupied by an application, user, or queue.
- **Dynamic update of configuration files**: Administrators can dynamically modify configuration parameters to manage clusters online.

5.3.4 Resource Allocation Model of Capacity Scheduler

Figure 5-8 shows the resource allocation model of Capacity Scheduler. This model consists of queues, applications, and containers. Capacity Scheduler maintains queue information. Users can submit applications to one or more queues. Each time NodeManager sends a heartbeat message, the scheduler selects a queue based on certain rules, selects an

application from the queue, and attempts to allocate resources to the application. The scheduler responds to the requests for local resources, intra-rack resources, and resources on any server in sequence.

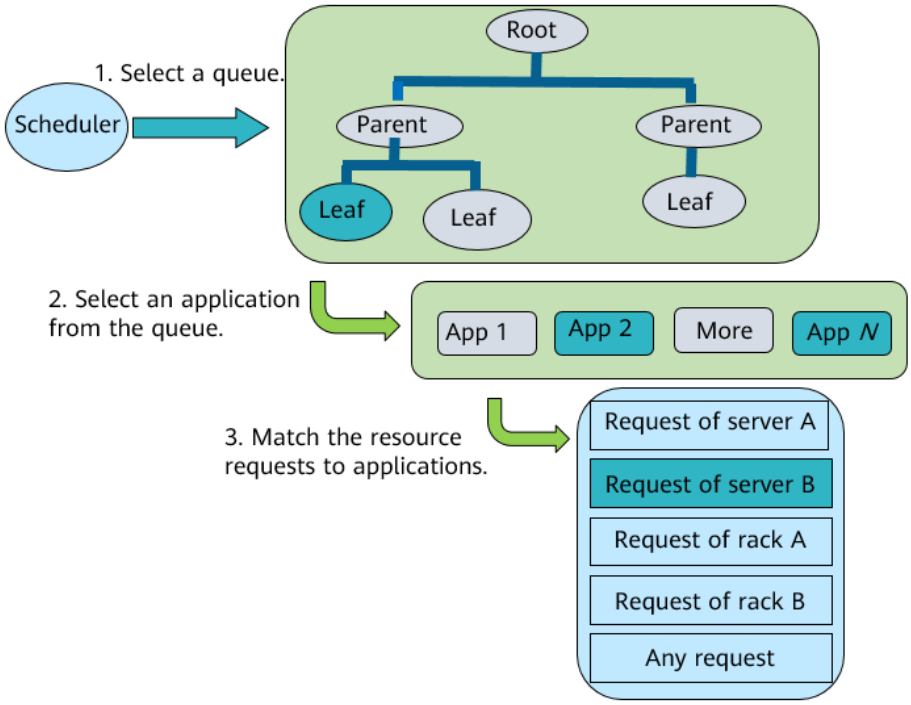


Figure 5-7 Resource allocation model of Capacity Scheduler

5.4 Enhanced Features

5.4.1 Dynamic Memory Management for YARN

In a cluster, multiple containers may be generated during task running. Previously, when the container capacity on a node exceeds the running memory size, NodeManager will stop the containers even though the overall memory usage is low, which results in job failures. This often causes job failures. After dynamic memory management is introduced, only when the total memory usage of all containers in a NodeManager exceeds the specified threshold, the containers consuming much memory are stopped. The NodeManager memory threshold (unit: GB) is calculated as follows: **yarn.nodemanager.resource.memory-mb** x 1,024 x **yarn.nodemanager.dynamic.memory.usage.threshold**. Dynamic memory management significantly improves the memory utilization of containers in a NodeManager.

5.4.2 Label-based Scheduling of YARN

Label-based scheduling is a scheduling policy that enables users to label each NodeManager, for example, labeling NodeManager with high-memory or high-IO. In the mean time, users can label each queue in the scheduler. That is, queues are bound with labels. Jobs submitted to a queue use resources on the labeled nodes. That is, tasks are running on labeled nodes.

Before label-based scheduling is used, it is impossible to locate the node to which a task is submitted. Tasks are allocated to nodes based on some algorithms and conditions. Label-based scheduling can specify the nodes to which tasks are submitted. For example,

applications that have demanding memory requirements may run on servers with standard performance. This results in low computing efficiency. Through label-based scheduling, tasks that consume a large amount of memory are submitted to queues bound to high-memory labels. In this way, tasks can run on high-memory machines, improving the cluster running efficiency.

5.5 Quiz

1. If no partitioner is defined, how is data partitioned before being sent to the reducer?
2. What are the differences between combine and merge?

6 Spark — In-memory Distributed Computing Engine & Flink — Stream and Batch Processing in a Single Engine

6.1 Spark — In-memory Distributed Computing Engine

6.1.1 Spark Overview

6.1.1.1 Introduction

Spark is a universal parallel computing framework developed by the UC Berkeley AMPLab in 2009, was open sourced in early 2010, and moved to the Apache Software Foundation in 2013. Today, the project has become a top open source project in Apache Software Foundation.

It is a fast, versatile, and scalable memory-based big data computing engine. MapReduce-based computing engines typically output intermediate results to disks for storage and fault tolerance. Spark stores intermediate results in memory to reduce the I/O of the underlying storage system and improve the computing speed. It is a one-stop solution that integrates batch processing, real-time stream processing, interactive query, graph computing, and machine learning.

Spark can process data requirements in different scenarios, including batch processing, interactive query, real-time stream processing, and machine learning. Spark not only outperforms MapReduce, but also is compatible with the Hadoop ecosystem. It can run on Hadoop HDFS to provide enhanced functions. To some extent, Spark replaces Hadoop MapReduce. It is still compatible with YARN and Apache Mesos in Hadoop so existing Hadoop users can easily migrate their workloads to Spark.

6.1.1.2 Typical Use Cases

- Batch processing can be used for extracting, transforming, and loading (ETL).
- Machine learning can be used to automatically determine whether the comments of online buyers are positive or negative.
- Interactive analysis can be used to query the Hive warehouse.
- Stream processing can be used for real-time businesses analysis (such as page-click streams), recommendation systems, and public opinion analysis.

6.1.1.3 Highlights

The highlights of Spark are as follows:

- **Lightweight**: Spark only has 30,000 lines of core code. It also supports the easy-to-read and rich Scala language.
- **Fast**: Spark can respond to small data set queries in just subseconds. Spark is faster than MapReduce, Hive or Pregel for iterative machine learning of large dataset applications, such as ad-hoc query and graph programming. Spark features in-memory computing, data locality, transmission optimization, and scheduling optimization.
- **Flexible**: Spark offers flexibility at different levels. Spark uses the Scala trait dynamic mixing policy (such as replaceable cluster scheduler and serialized library). Spark allows users to extend new data operators, data sources, and language bindings. Spark supports a variety of paradigms such as in-memory computing, multi-iteration batch processing, ad-hoc query, streaming processing, and graph programming.
- **Smart**: Spark seamlessly combines with Hadoop and is compatible with the Hadoop ecosystem. Graph computing uses Pregel and PowerGraph APIs as well as the point division idea of PowerGraph.

6.1.2 Spark Data Structure

6.1.2.1 RDD — Core Concept of Spark

Resilient Distributed Datasets (RDDs) are elastic, read-only, and partitioned distributed data sets. Spark abstracts data into RDDs. A RDD is a collection of data distributed on multiple nodes in a cluster. RDDs can be used to concurrently use distributed data in the cluster.

A RDD can be created in either of the following ways:

- Created from the Hadoop file system or other Hadoop-compatible storage systems, such as Hadoop Distributed File System (HDFS).
- Converted from a parent RDD.

RDD storage: Users can choose different storage levels (11 levels available) to store RDDs for reuse. By default, RDDs are stored in memory and overflow to disks if memory is insufficient. RDDs need to be partitioned to reduce network transmission costs and distribute computing. RDDs are partitioned according to each record key to effectively join two data sets.

RDDs have a lineage mechanism, which allows for rapid data recovery when data is lost.

6.1.2.2 RDD Dependencies

Spark lineage mechanism depending on the dependencies between RDDs ensures the system fault tolerance rate. The dependencies are classified into narrow dependency and wide dependency.

- Narrow dependency: The partitions of a parent RDD and its child RDDs are in one-to-one relationship. One or multiple partitions of a parent RDD correspond to one partition of each child RDD, and no shuffle is generated between them.

- Wide dependency: One partition of a parent RDD corresponds to multiple partitions of a child RDD. Data needs to be shuffled.

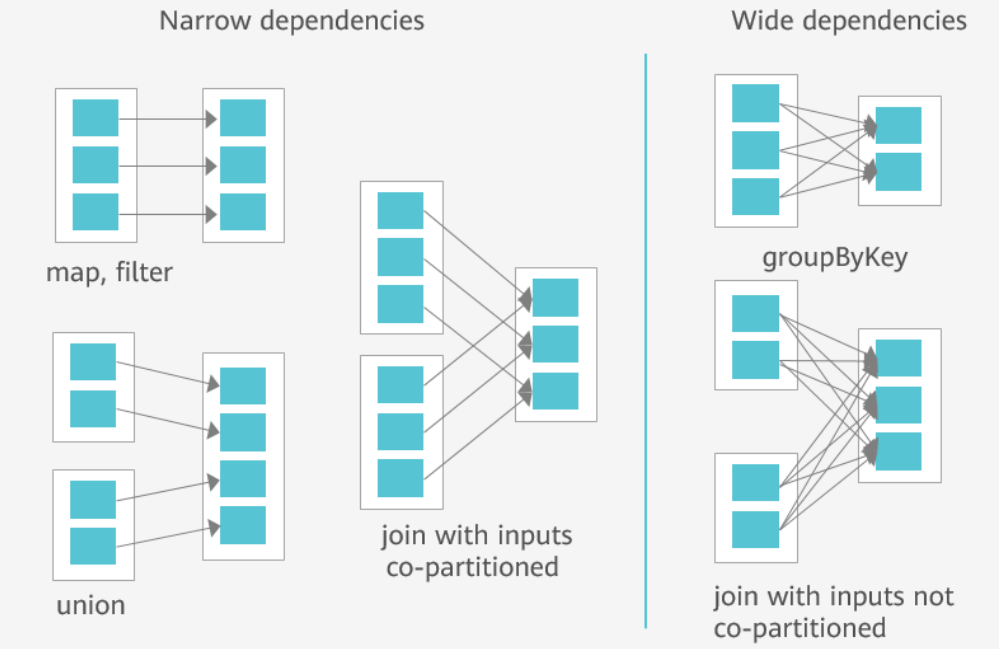


Figure 6-1 RDD dependencies

In the preceding figure, the left part shows narrow dependencies, while wide dependencies are displayed on the right.

Now let's learn about how RDD stages are divided. Spark tasks form a directed acyclic graph (DAG) based on the dependency between RDDs. The DAG is submitted to DAGScheduler. DAGScheduler divides the DAG into multiple stages that depend on each other based on the wide and narrow dependencies between RDDs. When a wide dependency occurs, stages are divided. Each stage contains one or more tasks. The tasks are then submitted to TaskScheduler as taskSets. An example RDD stage division is as follows:

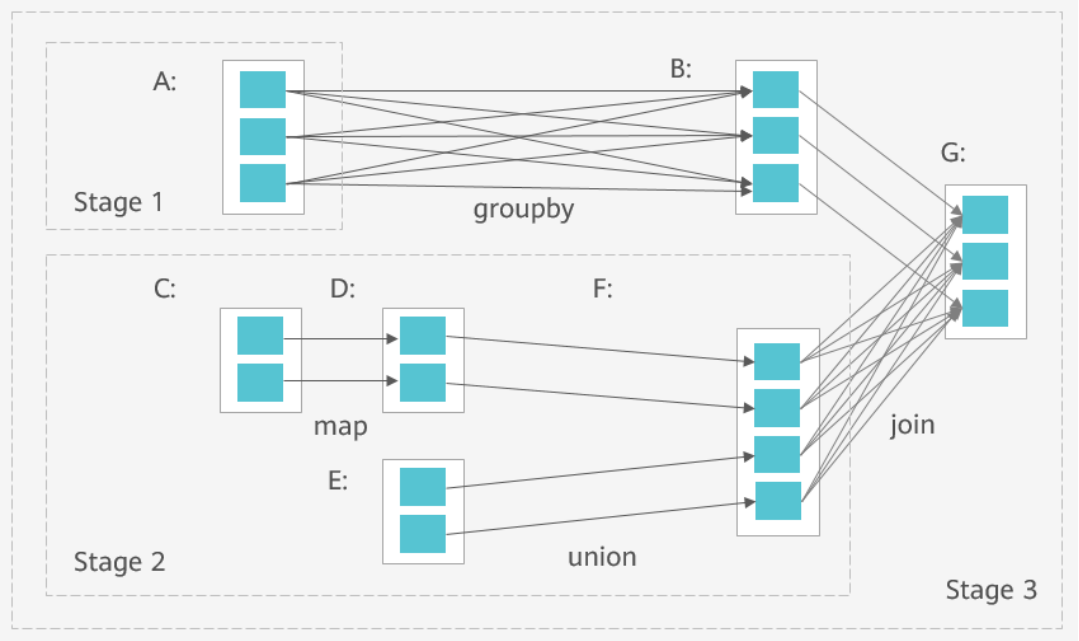


Figure 6-2 Example RDD stage division

6.1.2.3 Differences Between Wide and Narrow Dependencies — Operator

Spark provides a large number of operators to use RDDs. An operator can be considered as a method or function for using RDDs. There are two types of operators: transform operator and action operator. The transform operator converts one RDD into another. However, the transform operator does not trigger job submission, but only completes intermediate job processing. That is, the operation of converting one RDD to another is not executed immediately, but triggered only when the action operator is performed. The parameter type of the transform operator is value or key-value.

The action operator can be considered as an operator that performs non-conversion operations on RDDs, for example, persisting results to an external storage system. The action operator triggers SparkContext to submit jobs.

Generally speaking, the transform operator is used to transform RDDs, and its return value is RDDs. The action operator is used to perform non-transform operations on RDDs, and its return value is not RDDs. A Spark program can be executed only when there is at least one action operator. Otherwise, an exception is thrown.

The difference between narrow dependency and wide dependency is that narrow dependency indicates that the partition of a parent RDD can be used by at most one partition of a child RDD, for example, **map**, **filter**, and **union**. A wide dependency indicates that each partition of a parent RDD may be depended on by multiple partitions of a child RDD, for example, **groupByKey**, **reduceByKey**, and **sortByKey**.

6.1.2.4 Differences Between Wide and Narrow Dependencies — Fault Tolerance

If a node is faulty:

Narrow dependency: Only the parent RDD partition corresponding to the child RDD partition needs to be recalculated.

Wide dependency: In extreme cases, all parent RDD partitions need to be recalculated.

6.1.2.5 Differences Between Wide and Narrow Dependencies — Data Transmission

Wide dependencies usually require shuffle operations, and each partition of the parent RDD is depended on by multiple child RDD partitions. The execution may involve data transmission between multiple nodes. The partition of each parent RDD with narrow dependency is transferred to only one child RDD partition. Typically, the conversion can be completed on one node.

6.1.2.6 RDD Operation Type

Spark operations can be classified into creation, control, conversion, and behavior.

- Creation: used to create a RDD. A RDD is created through a memory collection and an external storage system, or by a transformation operation. The two types of RDD can be operated in the same way to obtain a series of extensions, such as sub-RDD, and form a lineage diagram.
- Control: RDD persistence is performed. A RDD can be stored in the disk or memory based on different storage policies. For example, the cache API caches the RDD in the memory by default. Typically, 60% of the memory of the execution node is used to cache data, and the remaining 40% is used to run tasks.
- Transformation: A RDD is transformed into a new RDD through certain operations. The transformation operation of the RDD is a lazy operation, which only defines a new RDD but does not execute it immediately. An example is as follows:

Table 6-1 Example transformation operation

Transformation	Description	Transformation
map(func)	Uses the **func** method to generate a new RDD for each element in the RDD that invokes **map**.	map(func)
filter(func)	**func** is used for each element of a RDD that invokes **filter** and then a RDD with elements containing **func** (the value is **true**) is returned.	filter(func)
reduceBykey(func,[numTasks])	It is similar to **groupBykey**. However, the value of each key is calculated based on	reduceBykey(func,[numTasks])

Transformation	Description	Transformation
	the provided **func** to obtain a new value.	
join(otherDataset,[numTasks])	If the data set is **(K, V)** and the associated data set is **(K, W)**, then **(K, (V, W))** is returned. **leftOuterJoin**, **rightOutJoin**, and **fullOuterJoin** are supported.	join(otherDataset,[numTasks])

- Control: RDD persistence is performed. A RDD can be stored in the disk or memory based on different storage policies. For example, the cache API caches the RDD in the memory by default. An example is as follows:

Table 6-2 Example control operation

Action	Description
reduce(func)	Aggregates elements in a dataset based on functions.
collect()	Used to encapsulate the **filter** result or a small enough result and return an array.
count()	Collects statistics on the number of elements in a RDD.
first()	Obtains the first element of a data set.
take(n)	Obtains the top elements of a dataset and returns an array.
saveAsTextFile(path)	Writes the data set to a text file or HDFS. Spark converts each record into a row and writes the row to the file.

6.1.2.7 DataFrame

Similar to RDD, DataFrame is also an invariable, elastic, and distributed data set. In addition to data, it also records data structure information, that is, schema. The schema is similar to a two-dimensional table. The query plan of DataFrame can be optimized using Spark Catalyst Optimiser. The optimized logical execution plan is still logical and cannot be understood by the Spark system. In this case, you need to convert a logical execution plan to a physical plan. In this way, a logically feasible execution plan is changed to a plan that Spark can actually execute.

6.1.2.8 DataSet

DataSet, a typed dataset, includes Dataset[Car] and Dataset[Person]. DataFrame is a special case of DataSet (**DataFrame=Dataset[Row]**). Therefore, you can use the **as** method to convert DataFrame to DataSet. **Row** is a common type where all table structure information is represented by row.

6.1.2.9 Differences Between DataFrame, DataSet, and RDD

DataFrame, DataSet, and RDD are elastic distributed data sets and support mutual conversion. They have many common functions, such as filter and sorting. All of them are lazy and will not be executed immediately during creation and conversion. The traversal operation starts only when the Action operator is used. The differences are as follows:

- DataFrame: (1) The type of each row in DataFrame is fixed to **Row**, and the value of each field can be obtained only through parsing. The value of each column cannot be directly accessed. (2) The DataFrame compiler lacks security check for types.
- DataSet: (1) The type of each row is not fixed, which can be **Person** or **Row**. (2) The DataSet type is more secure.
- RDD: (1) APIs can be used for Spark1.*X* modules. (2) Spark SQL operations are not supported. (3) Automatic code optimization is not supported.

6.1.3 Spark Principles and Architecture

6.1.3.1 Spark Ecosystem

As shown in the following figure, Spark Core is the core of the Spark ecosystem. Data is read from persistence layers such as HDFS and HBase, and Spark's standalone, YARN, and Mesos ClusterManager are used to schedule jobs for computing of Spark applications that can come from different components. For example, Spark Shell/Spark Submit is for batch processing, Spark Streaming is for real-time processing, Spark SQL is for ad hoc query, MLlib is for machine learning, GraphX is for graph processing, and SparkR is for mathematical computing.

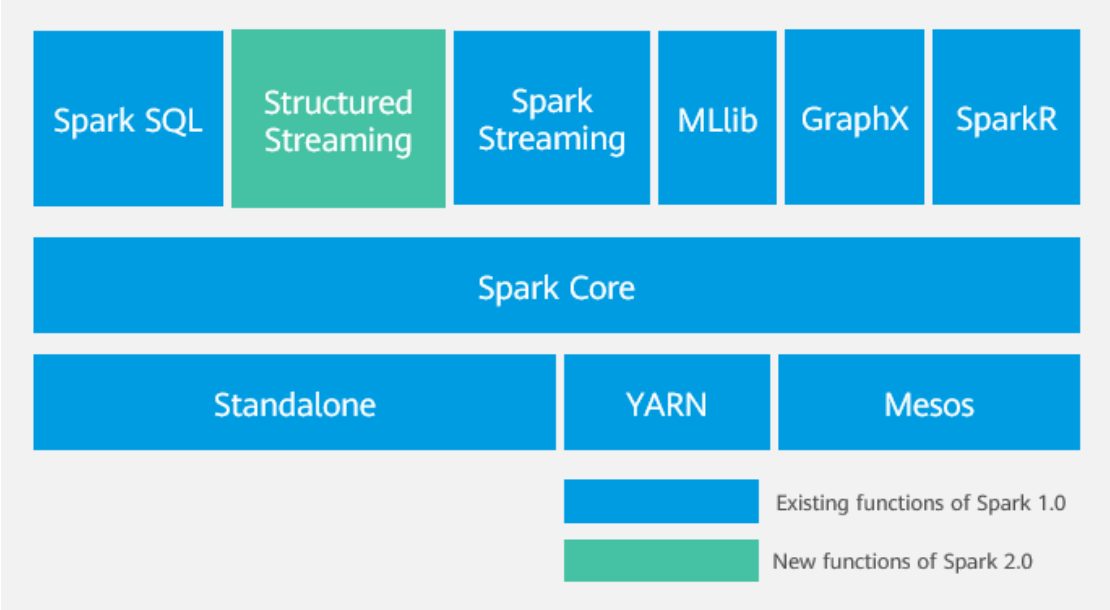


Figure 6-3 Spark ecosystem

Spark Core: It is similar to the distributed memory computing framework of MapReduce. The most striking feature of Spark Core is that it directly inputs the intermediate computing results into the memory to improve computing performance. In addition to the resource management framework in standalone mode, it also supports the resource management system of YARN and Mesos. FusionInsight integrates Spark on YARN. Other modes are currently not supported.

Spark SQL: It is a Spark component for processing structured data. As a part of the big data framework of Apache Spark, it is mainly used to process structured data and perform SQL-like data queries. With Spark SQL, users can perform ETL operations on data in different formats (such as JSON, Parquet, and ORC) and on different data sources (such as HDFS and databases), completing specific query operations.

Spark Streaming: It is the stream processing engine for mini batch processing. After stream data is fragmented, the computing engine of Spark Core is used to process the data. Compared to storm, it has slightly slower real-time performance but higher throughput.

Spark MLlib: It implements some common machine learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimension reduction, and underlying optimization, and the algorithms can be expanded. Spark MLlib lowers the barrier to the use of machine learning. Developers can work on machine learning as long as they have certain theoretical knowledge.

GraphX: GraphX is an API used for parallel graph computing in Spark. It can be considered as the rewriting and optimization of GraphLab and Pregel in Spark. Compared with other distributed graph computing frameworks, GraphX provides a one-stop data solution based on Spark, which can efficiently complete a complete set of graph computing pipelines.

Structured Streaming is unique to Spark 2.0 or later. It is a streaming data processing engine built on Spark SQL. You can write streaming computing processes just like you would with static RDD data. When streaming data is continuously generated, Spark SQL

processes the data incrementally and continuously and updates the results to the result set.

6.1.3.2 Spark SQL

Spark SQL allows developers to directly process RDDs and query external data stored in Hive and HBase. An important feature of Spark SQL is that it can process relational tables and RDDs in a unified manner so that developers can easily use SQL commands to perform external queries and more complex data analysis.

Spark SQL uses a similar method to relational databases to process SQL statements. SparkSQL parses SQL statements to form a tree, and then uses rules to bind and optimize the tree.

Lexical and syntax parsing: Parses the lexical and syntax of the read SQL statements to identify keywords (such as SELECT, FROM, and WHERE), expressions, projections, and data sources, and then determines whether the SQL statement is standard and forms a logical plan.

Bind: Spark SQL binds a SQL statement to a database's data dictionary (column, table, or view). If the related projection and data source exist, the statement can be executed.

Optimize: Spark SQL provides several execution plans and returns the data sets obtained from the database. **Execute**: Spark SQL executes the optimal execution plan obtained in the previous step and returns the datasets obtained from the database.

The differences between Spark SQL and Hive SQL are as follows:

- The execution engine of Spark SQL is Spark Core, while the default execution engine of Hive is MapReduce.
- Spark SQL executes 10 to 100 times faster than Hive.
- Spark SQL does not support buckets, but Hive does.

The relationship between Spark SQL and Hive SQL is as follows:

- Spark SQL depends on Hive's metadata.
- Spark SQL is compatible with most Hive syntax and functions.
- Spark SQL can use the custom functions of Hive.

6.1.3.3 Spark Structured Streaming

Spark Structured Streaming is a streaming data processing engine built on Spark SQL. You can write streaming computing processes just like you would with static RDD data. When streaming data is continuously generated, Spark SQL processes the data incrementally and continuously and updates the results to the result set. The core of Structured Streaming is to regard streaming data as a database table where data is continuously increasing. Such a data processing model is similar to data block processing. It can apply some query operations of the static database table to streaming data computing. Spark executes standard SQL query statements to obtain data from an unbounded table.

6.1.3.4 Spark Streaming

Spark Streaming is a stream processing system that performs high-throughput and fault-tolerant processing on real-time data streams. It can perform complex operations such as map, reduce, and join on multiple data sources (such as Kafka, Flume, Twitter, Zero, and

TCP sockets) and save the results to external file systems, databases, or real-time dashboards.

The core idea of Spark Streaming is to split stream computing into a series of short batch jobs. The batch processing engine is Spark Core. That is, the input data of Spark Streaming is divided into segments based on a specified time slice (for example, 1 second), each segment is converted into RDDs in Spark, then the DStream conversion in Spark Streaming is transformed to the RDD conversion in Spark. As a result, the intermediate results of RDD conversion are saved in the memory.

Storm is a well-known framework in the real-time computing field. Compared with Storm, Spark Streaming provides higher throughput. They have better performance than each other in different scenarios.

Use cases of Storm:

- Storm is recommended when even a one-second delay is unacceptable. For example, a financial system requires real-time financial transaction and analysis.
- If a reliable transaction mechanism and reliability mechanism are required for real-time computing, that is, data processing must be accurate, Storm is ideal.
- If dynamic adjustment of real-time computing program parallelism is required based on the peak and off-peak hours, Storm can maximize the use of cluster resources (usually in small companies with resource constraints).
- If SQL interactive query operations and complex transformation operators do not need to be executed on a big data application system that requires real-time computing, Storm is preferred.

If real-time computing, a strong transaction mechanism, and dynamic parallelism adjustment are not required, Spark Streaming should be considered. Located in the Spark ecological technology stack, Spark Streaming can seamlessly integrate with Spark Core and Spark SQL. That is, delay batch processing, interactive query, and other operations can be performed immediately and seamlessly on immediate data that is processed in real time. This feature significantly enhances the advantages and functions of Spark Streaming.

6.2 Flink — Stream and Batch Processing in a Single Engine

6.2.1 Flink Principles and Architecture

6.2.1.1 Core Concepts of Flink

Apache Flink is an open source stream processing framework for distributed, high-performance stream processing applications. Flink is a distributed processing engine that performs stateful computing on unbounded and bounded data streams. It can run in all common cluster environments and perform computing at memory speed and any scale. Flink is the only distributed stream data processing framework that features high throughput, low latency, and high performance in the open source community.

The biggest difference between Flink and other stream computing engines is state management.

What is a state? For example, when a stream computing system or task is developed for data processing, data statistics such as Sum, Count, Min, or Max need to be collected. These values need to be stored. These values or variables can be understood as a state because they need to be updated continuously. If the data sources are Kafka and RocketMQ, the read location and offset may need to be recorded. These offset variables are the states to be calculated.

Flink provides built-in state management. You can store states in Flink instead of storing them in an external system. This:

- Reduces the dependency of the computing engine on external systems, simplifying deployment and O&M.
- Significantly improves performance.

If Redis or HBase wants to access the states in Flink, it must access the states via the Internet or RPC. If the states are accessed through Flink, they are accessed only through its own process. In addition, Flink periodically takes state checkpoints and stores them to a distributed persistence system, such as HDFS. In case of a failure, Flink resets its state to the last successful checkpoint and continues to process the stream. There is no impact on user data.

6.2.1.2 Flink Runtime Architecture

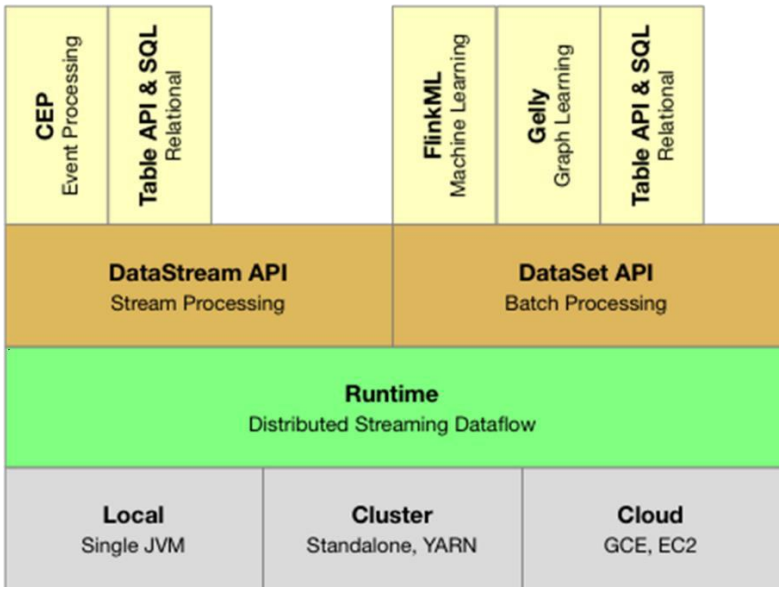


Figure 6-4 Flink runtime architecture

Flink is a system with a hierarchical architecture. It is divided into three layers. Each layer provides specific abstractions to serve upper-layer components. In terms of deployment, YARN can be deployed on a single node, cluster, or cloud. Typically, YARN is deployed in a cluster. At the core layer, there is a distributed streaming data processing engine. At the API layer, there are stream processing APIs and batch processing APIs. Stream processing supports event processing and table operations. Batch processing supports machine learning, graph computing, and table operations.

Flink provides the following deployment plans: **Local**, which indicates local deployment, **Cluster**, which indicates cluster deployment, and **Cloud**, which indicates cloud deployment.

The runtime layer is a common engine for stream processing and batch processing of Flink, receiving applications in a JobGraph. A JobGraph is a general parallel data flow, which has more or fewer tasks to receive and generate data streams.

Both the DataStream API and DataSet API can generate JobGraphs using a specific compiling method. The DataSet API uses the optimizer to determine the application optimization method, while the DataStream API uses the stream builder to perform this task.

Table API supports query of structured data. Structured data is abstracted into a relationship table. Users can perform various query operations on the relationship table through SQL-like DSL provided by Flink. Java and Scala are supported.

The Libraries layer corresponds to some functions of different Flink APIs, including the table for processing logical table query, FlinkML for machine learning, Gelly for image processing, and CEP for complex event processing.

6.2.1.3 DataStream

Flink uses class DataStream to represent the stream data in Flink programs. You can think of DataStream as immutable collections of data that can contain duplicates. The number of elements in DataStream is unlimited.

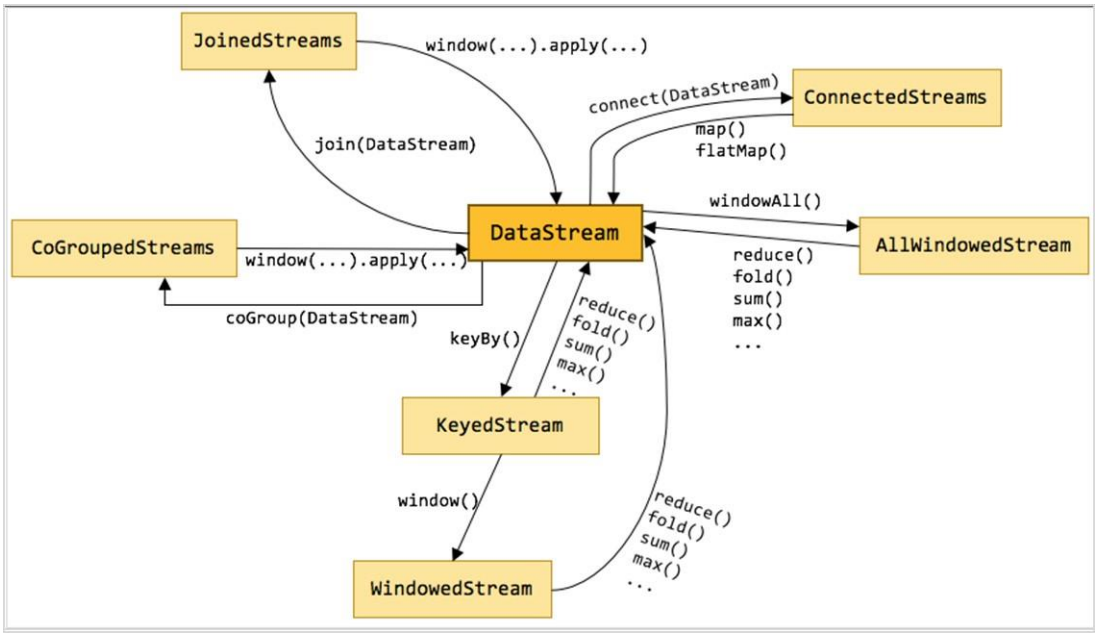


Figure 6-5 Main operators of DataStream

The operator operations between DataStreams are described as follows:

- Window operations are associated using the **reduce**, **fold**, **sum** and **max** functions.
- ConnectedStream: Connects streams using the **flatmap** and **map** functions.
- JoinedStream: Performs the join operation between streams using the **join** function, which is similar to the join operation in a database.

- CoGroupedStream: Associates streams using the **coGroup** function, which is similar to the group operation in a relational database.
- KeyedStream: Processes data flows based on keys using the **keyBy** function.

6.2.1.4 DataSet

DataSet programs in Flink are regular programs that transform data sets (for example, filtering, mapping, joining, and grouping). The datasets are initially created by reading files or from local collections. Results are returned via sinks, which can write the data to (distributed) files or to standard output (for example, the command line terminal).

6.2.1.5 Flink Program

Flink programs consist of three parts: source, transformation, and sink. The source reads data from data sources such as HDFS, Kafka, and text. The transformation is responsible for data transformation operations. The sink outputs final data (such as HDFS, Kafka, and text). Data that flows between these parts is called a stream.

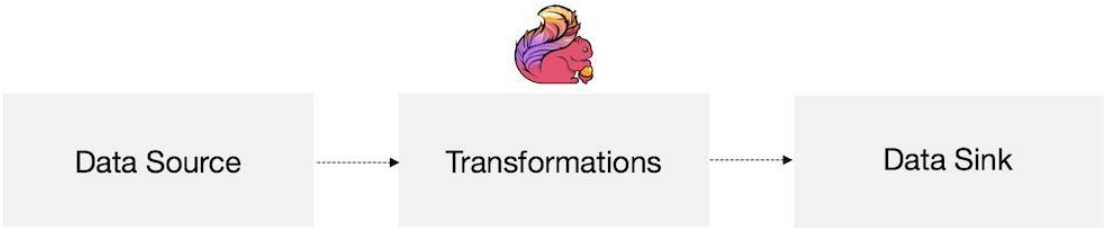


Figure 6-6 Flink program composition

6.2.1.6 Flink Program Composition

- Obtain the execution environment, StreamExecutionEnvironment, which is the basis of all Flink programs.
  - StreamExecutionEnvironment.getExecutionEnvironment
  - StreamExecutionEnvironment.createLocalEnvironment
  - StreamExecutionEnvironment.createRemoteEnvironment
- Load/create the raw data.
- Specify transformations on this data.
- Specify where to put the results of your computations.
- Trigger the program execution.

6.2.1.7 Flink Data Sources

Batch processing data sources

- File-based data. The file formats include HDFS, Text, CSV, Avro, and Hadoop input format.
- JDBC-based data
- HBase-based data
- Collection-based data

Stream processing data sources

- File-based data: **readTextFile(path)** reads a text file line by line based on the TextInputFormat read rule and returns the result.
- Collection-based data: **romCollection()** creates a data flow using a collection. All elements in the collection must be of the same type.
- Queue-based data: Data in message queues such as Kafka and RabbitMQ is used as the data source.
- User-defined source: Data sources are defined by implementing the SourceFunction API.

6.2.1.8 Flink Program Running

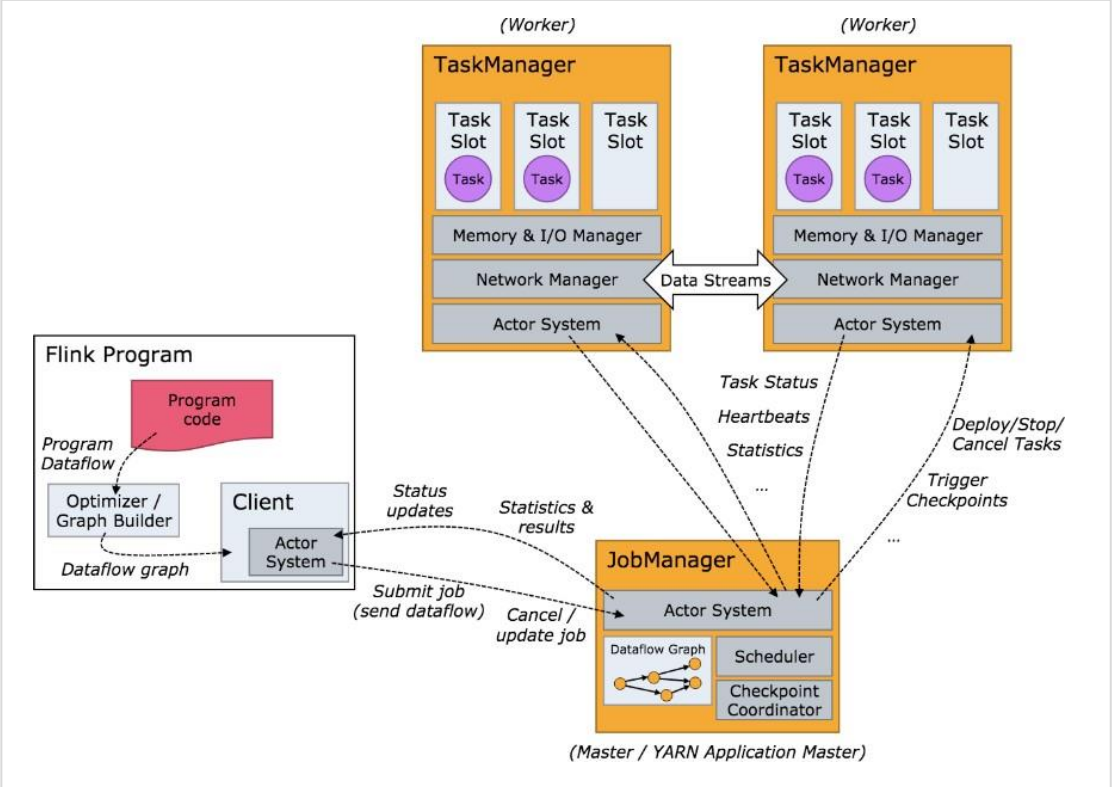


Figure 6-7 Flink program running structure

Flink adopts a master-slave architecture. If a Flink cluster is started, a JobManager process and at least one TaskManager process are also started. After a Flink program is submitted, a client is created for preprocessing. The program is converted into a DAG that indicates a complete job and is submitted to JobManager. JobManager assigns each task in the job to TaskManager. The compute resources in Flink are defined by task slots. Each task slot represents a fixed-size resource pool of TaskManager. For example, a TaskManager with three slots evenly divides the memory managed by the TaskManager into three parts and allocates them to each slot. Slotting resources means that tasks from different jobs do not compete for memory. Currently, slots support only memory isolation but not CPU isolation.

When a Flink program is executed, it is first converted into a streaming dataflow, which is a DAG consisting of a group of streams and transformation operators.

A user submits a Flink program to JobClient. JobClient processes, parses, and optimizes the program, and then submits the program to JobManager. Then, TaskManager runs the task.

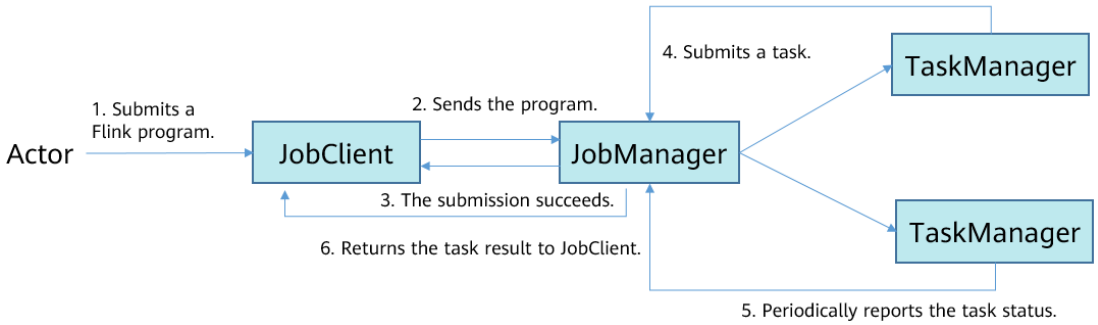


Figure 6-8 Flink job running process

Client: The Flink client is used to submit jobs (streaming jobs) to Flink. TaskManager: TaskManager is a service execution node of Flink, which executes specific tasks. A Flink system can have multiple TaskManagers that are all equivalent to each other.

JobManager: JobManager is a management node of Flink that manages all TaskManagers and schedules tasks submitted by users to specific TaskManagers. In high availability (HA) mode, multiple JobManagers are deployed. Among these JobManagers, one is elected as the leader, and the others are on standby.

TaskSlot: TaskSlot is similar to a container in Yarn, used for resource isolation. This component contains memory resources and does not contain CPU resources. Assume that each TaskManager contains three task slots, which means that each can perform a maximum of three tasks at a time. The task slot has dedicated memory, so multiple jobs do not affect each other. Task slots can share JVM resources, datasets, data structures, and TCP connections and heartbeat messages using Multiplexing.

Task is the task execution unit.

6.2.1.9 Complete Flink Program

```
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment; import
org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.api.common.functions.FilterFunction; public class Example {
public static void main(String[] args) throws Exception { final StreamExecutionEnvironment env
=StreamExecutionEnvironment.getExecutionEnvironment();
DataStream<Person> flintstones = env.fromElements(
new Person("Fred", 35), new Person("Wilma", 35), new Person("Pebbles", 2)); DataStream<Person>
adults = flintstones.filter(new FilterFunction<Person>() {
@Override
public boolean filter(Person person) throws Exception { return person.age >= 18;}
});
adults.print(); env.execute();
}
public static class Person { public String name; public Integer age; public Person() {};
public Person(String name, Integer age) { this.name = name;
this.age = age;
};
public String toString() {
return this.name.toString() + ": age" + this.age.toString();
```

```
};
}
}
```

6.2.1.10 Flink Data Processing

Flink can process both bounded and unbounded data streams.

Bounded data stream and batch processing: Bounded data streams have a clear start and end. Bounded streams can be computed after all data is obtained. All data in a bounded stream can be sorted, without ingesting events in an order. Bounded stream processing is often referred to as batch processing.

Unbounded data stream and stream processing: An unbounded data stream has a start but no end. It continuously generates data and needs to be processed continuously. Data needs to be processed immediately after it is obtained because the input is unbounded. Processing unbounded stream data typically requires ingesting events in a particular sequence, such as the sequence of the events that occurred, so that the integrity of the results can be inferred. Unbounded stream processing is often referred to as stream processing.

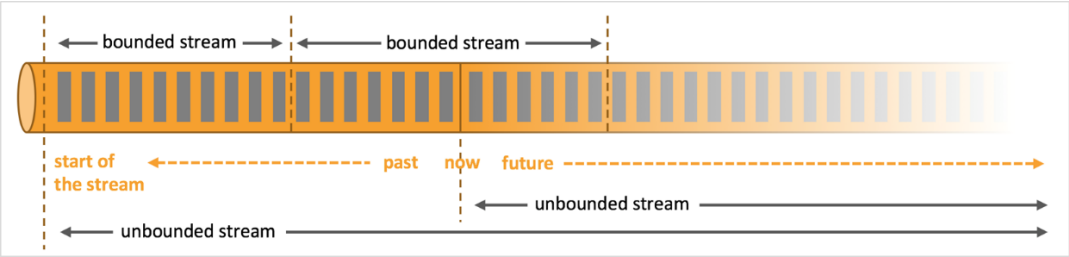


Figure 6-9 Bounded and unbounded streams

Batch processing and stream processing are two common data processing methods in big data analysis systems. **Batch processing** applies to bounded, persistent, and large amount of data. It is suitable for computing that requires access to a complete set of records and is typically used for offline statistics. **Stream processing** applies to unbounded and real-time data. Instead of performing operations on the entire data set, it performs operations on each data item transmitted through the system and is typically used for real-time statistics.

Streaming computing is classified into stateless computing and stateful computing. Stateless computing observes each independent event, does not store any status information or specific configuration, and the computing result is not used for later computing. After the computing is complete, the result is output and the next event is processed. Storm uses the stateless computing framework. Stateful computing stores intermediate result data of an operator in the memory or file system. After the next event arrives, the operator does computation based on the input data and the result from the previous state to output the result, rather than compute based on all raw data. This improves system performance and reduces resource consumption during data computing.

In Flink, everything is a continuous data stream. Offline data is a bounded stream, and real-time data is an unbounded stream. Flink is an open source computing platform for distributed data stream processing. It provides batch processing and stream processing APIs based on the same Flink runtime.

6.2.1.11 Example Batch Processing

Batch processing is a very special case of stream processing. Instead of defining a sliding or tumbling window over the data and producing results every time the window slides, we define a global window, with all records belonging to the same window.

```
val counts = visits
.keyBy("region")
.timeWindow(Time.hours(1))
.sum("visits")
```

For example, a simple Flink program that counts visitors at a website every hour, grouped by region continuously, is the following:

If you know that the input data is bounded, you can implement batch processing using the following code:

```
val counts = visits
.keyBy("region")
.window(GlobalWindows.create)
.trigger(EndOfTimeTrigger.create)
.sum("visits")
```

If the input data is bounded, the result of the following code is the same as that of the preceding code:

```
val counts = visits
.groupBy("region")
.sum("visits")
```

6.2.1.12 Flink Batch Processing Model

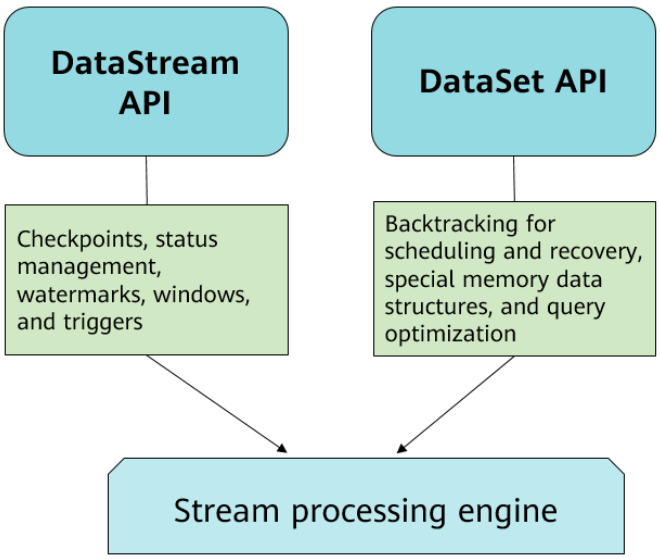


Figure 6-10 Flink underlying engine

Based on the stream processing engine, Flink has the following mechanisms:

Checkpoint and state mechanisms: used to implement fault tolerance and stateful processing. Watermark mechanism: used to implement the event clock.

Window and trigger: used to limit the calculation scope and define the time when the results are displayed.

On the same stream processing engine, Flink has another mechanism to implement efficient batch processing.

Backtracking for scheduling and recovery: introduced by Microsoft Dryad and now used in almost all batch processors.

Special memory data structure used for hashing and sorting: Part of the data can be allowed to flow from the memory to the hard disk when necessary.

Optimizer: shortens the time for generating results as much as possible.

The two sets of Flink mechanisms correspond to their respective APIs (DataStream API and DataSet API). When creating a Flink job, you cannot combine the two sets of mechanisms to use all Flink functions at the same time.

Flink supports two types of relational APIs: Table API and SQL. Both of these APIs are used for unified batch and stream processing, which means that relational APIs execute queries with the same semantics and produce the same results on unbounded real-time data streams and bounded historical data streams.

The Table API and SQL are becoming the main APIs to be used with unified stream and batch processing for analytical use cases. The DataStream API is the primary API for data-driven applications and pipelines.

6.2.2 Flink Time and Window

6.2.2.1 Time Background

For stream data processing programs, the most important characteristic is that data has time attributes. In stream processor programming, the processing of time is critical. For example, event stream data (such as server log data, web page click data, and transaction data) is continuously generated. In this case, you need to use keys to group events and count the events corresponding to each key at a specified interval. This is our known "big data" application.

6.2.2.2 Time Classification in Stream Processing

Flink supports different time concepts. Based on the location where time is generated, time is classified into processing time, event time, and ingestion time.

- **Event time**: time when an event occurs
- **Ingestion time**: time when an event arrives at the stream processing system
- **Processing time**: time when an event is processed by the system

In the actual situation, the sequence of events is different from the system time. These differences are caused by factors such as the network latency and processing time. An example is as follows:

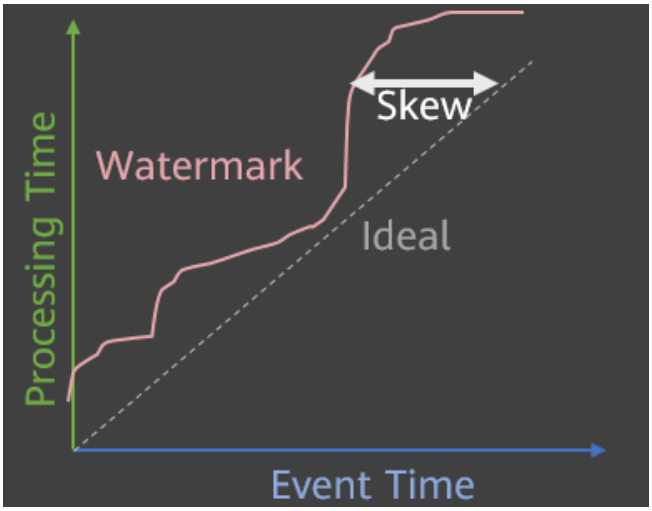


Figure 6-11 Differences among three time classifications

In this figure, the horizontal coordinate indicates the event time, and the vertical coordinate indicates the processing time. Ideally, the coordinate formed by the event time and processing time should form a line with a tilt angle of 45 degrees. However, the processing time is later than the event time. As a result, the sequence of events is inconsistent.

By default, Flink uses the processing time. To use the event time or ingestion time, call the **setStreamTimeCharacteristic()** method in the created **StreamExecutionEnvironment** to set the system time concept. In Flink stream processing, most services use event time. Processing time or ingestion time is used only when event time is unavailable. To use the event time, introduce its time attribute as follows:

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
// Add the time feature env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime) to each stream created by env from the calling time.
```

6.2.2.3 Time Semantics Supported by Flink

Processing Time	Event Time (Row Time)
Time in the real world	Time in the data world
Local time of the data node	Timestamp carried in a record
Simple processing	Complex processing
Uncertain results (cannot be reproduced)	Certain results (reproducible)

Figure 6-12 Differences between the processing time and event time

For most streaming applications, it is valuable to have the ability to reprocess historical data and produce consistent results with certainty using the same code used to process real-time data.

It is also critical to note the sequence in which events occur, not the sequence in which they are processed, and the capability to infer when a set of events are (or should be) completed. For example, consider a series of events involved in e-commerce transactions or financial transactions. These requirements for timely stream processing can be met by using the event timestamp recorded in the data stream instead of using the clock of the machine that processes the data.

6.2.2.4 Window Overview

Streaming computing system is a data processing engine designed to process infinite data sets. Infinite data sets refer to constantly growing data sets. Window is a method for splitting infinite data sets into finite blocks for processing. Window is the core idea of processing infinite streams. Streams are divided into different windows according to a fixed time. Then aggregation operation is performed on the data, so as to obtain statistical results in a specific time range.

6.2.2.5 Window Types

There are the following types of Flink windows:

- Keyed window and global window

  Flink generates two types of windows based on whether the data set is of the KeyedStream type:

  Keyed window: If the data set is of the KeyedStream type, the **window()** method of the DataStream API is invoked. Data is parallelly calculated in different task instances based on the key, and you can obtain statistical results of each key.

  Global window: If the data set is of the Non-Keyed type, the **windowsAll()** method is invoked. All data is calculated in a task and you can obtain global statistical results.

- Time window and count window

  Flink supports time window and count window based on service requirements. The count window is seldom used and is not described.

6.2.2.6 Time Window Types

Time windows are classified into three types based on service scenarios: tumbling window, sliding window, and session window.

- Tumbling window

  The window is divided based on the fixed time. The time points between windows do not overlap. The window is simple, and only the window length needs to be specified.

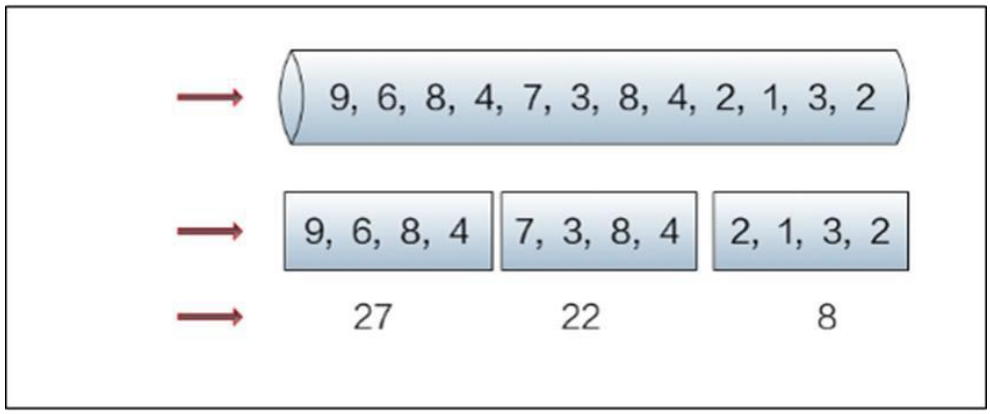


Figure 6-13 Tumbling window

- Sliding window

  The sliding window is common. It is created by adding sliding on the tumbling window. Time points between windows overlap. After the window size is fixed, the sliding window slides forward based on the preset sliding value, which is different from the tumbling window that moves forward based on the window size. The window size and sliding value determine the size of data overlapping between windows. If the sliding value is smaller than the window size, window overlapping occurs. If the sliding value is greater than the window size, windows are discontinuous, and some data may not be calculated in any window. When the sliding value is equal to the window size, the sliding window becomes a scrolling window. As shown in the following figure, the one-minute sliding window counts the values in the last one minute, but slides and emits the count every half minute.

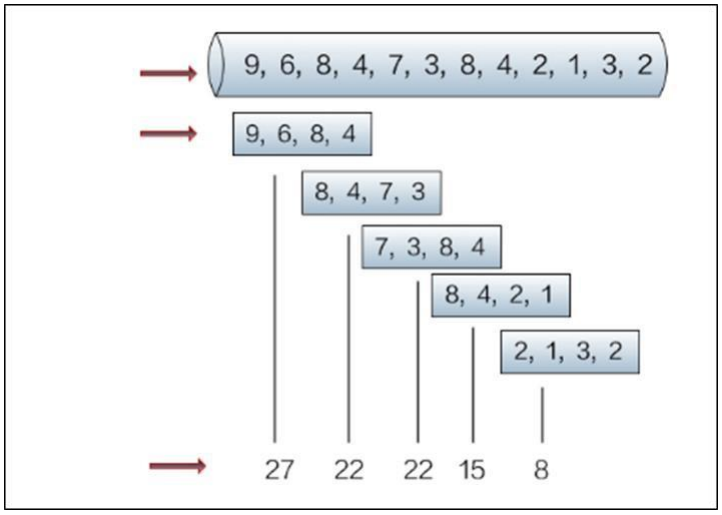


Figure 6-14 Sliding window

In the first sliding window, the values 9, 6, 8, and 4 are summed up, yielding the result 27. Next, the window slides by a half minute and the values 8, 4, 7, and 3 are summed up, yielding the result 22, etc.

- Session Window

  The session window aggregates the data with high activity in a certain period into a window for calculation. The window is triggered by the session gap. If no data is

received within the specified time, the window ends and the window calculation is triggered. Different from the tumbling window and sliding window, the session window does not require a fixed sliding value or window size. You only need to set the session gap for triggering window calculation and specify the upper limit of the inactive data duration.

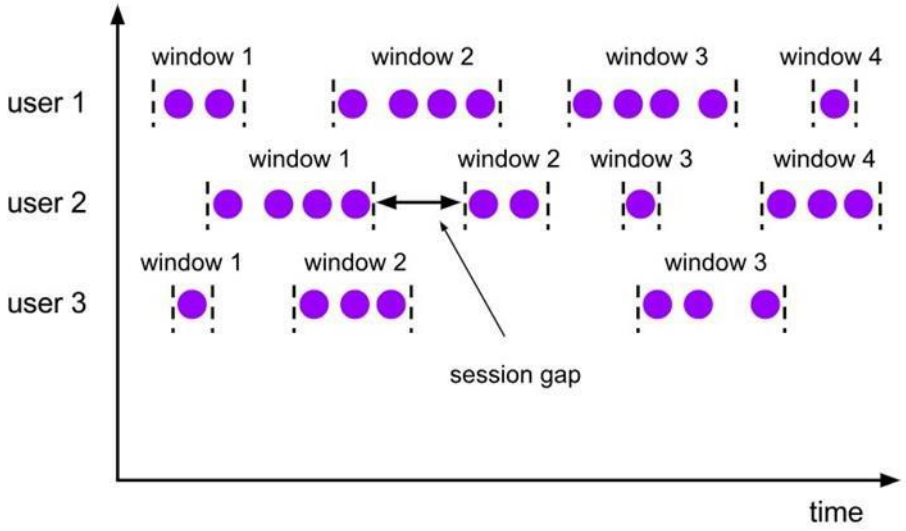


Figure 6-15 Session window

6.2.2.7 Code Definition

A tumbling time window of 1 minute can be defined in Flink simply as:

```
stream.timeWindow(Time.minutes(1))
```

A sliding time window of 1 minute that slides every 30 seconds can be defined as simply as:

```
stream.timeWindow(Time.minutes(1),Time.seconds(30))
```

To define window division rules, you can use the SessionWindows WindowAssigner API provided by Flink. If you've used SlidingEventTimeWindows or TumlingProcessingTimeWindows, you'll be familiar with this API.

```
DataStream input = ... DataStream result = input
.keyBy(<key selector>)
.window(SessionWindows.withGap(Time.seconds(<seconds>))
.apply(<window function>) // or reduce() or fold()
```

In this way, Flink automatically places elements in different session windows based on the timestamps of the elements. If the timestamp interval between two elements is less than the session gap, the two elements are in the same session. If the interval between two elements is greater than the session gap and no element can fill in the gap, the two elements are placed in different sessions.

6.2.3 Flink Watermark

6.2.3.1 Out-of-Order Data Problem

When using the event time to process stream data, you may encounter out-of-order data. Stream processing takes a certain period of time when an event is generated, arrives at a source, and is transmitted to an operator. In most cases, data transmitted to an operator is sorted based on the time sequence of events. However, the data may be out of order due to network delay. Out-of-order indicates that the sequence of events received by Flink is not strictly based on the event time.

6.2.3.2 Out-of-Order Data Example

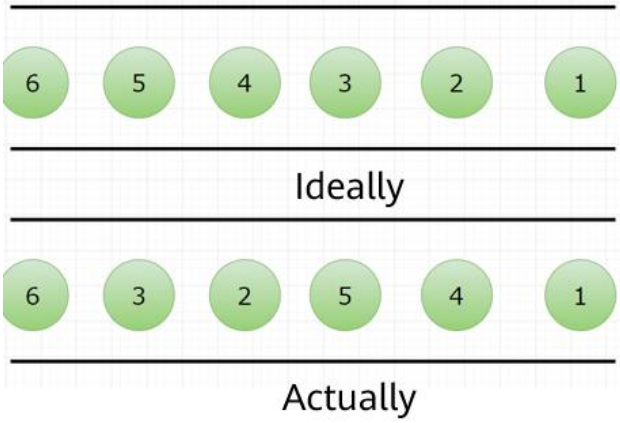


Figure 6-16 Out-of-order data example

There is a question. Once an out-of-order event occurs, if window running is determined only based on the event time, it cannot be determined whether all data is ready, but the system cannot wait indefinitely for data to arrive. In this case, a mechanism is required to ensure that the window is triggered for calculation after a specific time. This mechanism is called watermark.

Watermark is a mechanism for measuring the progress of event time. It is a hidden attribute of data and data carries watermarks. Watermarks are used to process out-of-order events and are usually implemented together with windows.

6.2.4 Watermark Principles

During the window processing of Flink, operations (such as grouping) can be performed on the data in the window only when all data arrives. If not all data is received, the processing can be started only after all data in the window is received. In this case, the watermark mechanism is required. Watermarks can measure the progress of data processing (showing the integrity of data arrival) and ensure that all event data arrives at Flink. In addition, correct and continuous results can be calculated based on the expectation when out-of-order or delayed arrival occurs. Watermark can be understood as a delayed trigger mechanism.

So how does Flink calculates the watermark value?

Watermark = Maximum event time in Flink (maxEventTime) – Specified delay time (t)

How does a window with watermarks trigger a window function?

The following figure shows the watermark of ordered streams (Watermark is set to 0).

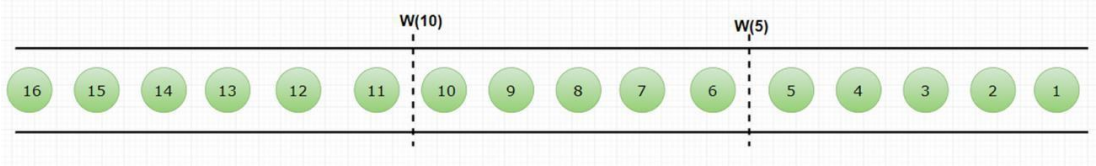


Figure 6-17 Ordered stream

The following figure shows the watermark of out-of-order streams (Watermark is set to 2).

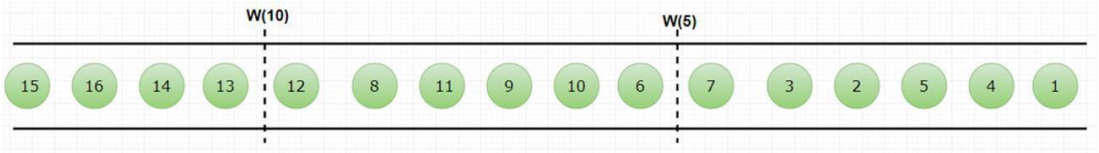


Figure 6-18 Out-of-order stream

When Flink receives each piece of data, a watermark is generated. The watermark is equal to the value of **maxEventTime** minus the delay. Once the watermark carried with the data is later than the stop time of the window that is not triggered, the corresponding window is triggered. Watermarks are carried with data. Therefore, if new data cannot be obtained during running, windows that are not triggered will never be triggered.

The allowed maximum delay is 2s. Therefore, the watermark of the event whose timestamp is 7s is 5s, and the watermark of the event whose timestamp is 12s is 10s. If window 1 is 1s to 5s and window 2 is 6s to 10s, window 1 is triggered when the event whose timestamp is 7s arrives and window 2 is triggered when the event whose timestamp is 12s arrives.

6.2.4.2 Delayed Data Processing

When the EventTime-based window processes streaming data, the watermark mechanism can only solve the out-of-order problem to a certain extent. In some cases, the data delay may be severe. Even watermarks cannot ensure that data is processed until all data arrives at the window. By default, Flink discards the delayed data. However, sometimes users expect that the data can be processed properly and the result can be output even if the data is delayed. For this reason, stream data programs explicitly specify some delay elements. A delay element refers to an element whose system time is later than the delay element timestamp. You can specify the maximum delay time (**0** by default) in the window using the following code:

```
input.keyBy(<keyselector>)
.window(<windowassigner>)
. allowedLateness(<time>)
.<windowed transformation>(<window function>);
```

6.2.4.3 Delayed Data Processing Mechanism

Delayed events are special out-of-order events. Different from common out-of-order events, their out-of-order degree exceeds the degree that watermarks can predict. As a

result, the window is closed before they arrive. In this case, you can use any of the following methods to solve the problem:

- Reactivate the closed windows and recalculate to correct the results (with the Side Output mechanism).
- Collect the delayed events and process them separately (with the Allowed Lateness mechanism).
- Consider delayed events as error messages and discard them.

By default, Flink uses the third method. The other two methods use the Side Output and Allowed Lateness mechanisms, respectively.

6.2.4.4 Side Output Mechanism

In the Side Output mechanism, a delayed event can be placed in an independent data stream, which can be used as a by-product of the window calculation result so that users can obtain and perform special processing on the delayed event. After allowedLateness is set, late data can also trigger the window for output. We can obtain this late data using the Side Output mechanism of Flink.

6.2.4.5 Allowed Lateness Mechanism

Allowed Lateness allows users to specify a maximum allowed lateness. After the window is closed, Flink keeps the state of windows until their allowed lateness expires. During this period, the delayed events are not discarded, but window recalculation is triggered by default. Keeping the window state requires extra memory. If the **ProcessWindowFunction** API is used for window calculation, each delayed event may trigger a full calculation of the window, which costs a lot. Therefore, the allowed lateness should not be too long, and the number of delayed events should not be too many.

6.2.5 Flink Fault Tolerance Mechanism

Flink is a stateful analysis engine by default. However, if a task is suspended during processing, its status in the memory will be lost. If the intermediate computing status is not stored, the computing task is restarted and all data needs to be recalculated. If an intermediate state is stored, you can restore the task to the intermediate state and continue to execute the task. For this, Flink introduces the checkpoint mechanism.

6.2.5.1 Checkpointing

Flink provides a checkpoint fault tolerance mechanism to ensure exactly-once semantics. Note that it can only ensure the exactly-once of the built-in operators of Flink. For the source and sink, if exactly-once needs to be ensured, these components themselves should support this semantics.

Flink provides a checkpoint fault tolerance mechanism based on the asynchronous, lightweight, and distributed snapshot technology. The snapshot technology allows you to take global snapshots of task or operator state data at the same point in time. Flink periodically generates checkpoint barriers on the input data set and divides the data within the interval to the corresponding checkpoints through barriers. When an exception occurs in an application, the operator can restore the status of all operators from the previous snapshot to ensure data consistency.

For applications with small state, these snapshots are very light-weight and can be taken frequently without impacting the performance much. During checkpointing, the state is stored at a configurable place (such as the JobManager node or HDFS).

6.2.5.2 Checkpointing Configuration

- Enabling checkpointing

  By default, checkpointing is disabled. To enable checkpointing, call **enableCheckpointing(n)**, where *n* is the checkpoint interval in milliseconds.

```
env.enableCheckpointing(1000) //Enable checkpointing and set the checkpointing interval to 1000 ms. Set this parameter based on site requirements. If the state is large, set it to a larger value.
```

- Exactly-once or at-least-once

  Exactly-once ensures end-to-end data consistency, prevents data loss and duplicates, and delivers poor Flink performance.

  At-least-once applies to scenarios that have high requirements on the latency and throughput but low requirements on data consistency.

  Exactly-once is used by default. You can use the **setCheckpointingMode()** method to set the semantic mode.

```
env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
```

- Checkpointing timeout

  Specifies the timeout period of checkpoint execution. Once the threshold is reached, Flink interrupts the checkpoint process and regards it as timeout.

  This metric can be set using the **setCheckpointTimeout** method. The default value is 10 minutes.

```
env.getCheckpointConfig().setCheckpointingTimeout(60000)
```

- Minimum interval between checkpoints

  When checkpoints frequently end up taking longer than the base interval because state grew larger than planned, the system constantly takes checkpoints. This can mean that too many computing resources are constantly tied up in checkpointing, thereby affecting the overall application performance. To prevent such a situation, applications can define a minimum duration between two checkpoints:

```
Env.getCheckpointConfig().setMinPauseBetweenCheckpoints(500)
```

- Maximum number of concurrent checkpoints

  The number of checkpoints that can be executed at the same time. By default, the system will not trigger another checkpoint while one is still in progress. If you configure multiple checkpoints, the system will trigger multiple checkpoints at the same time.

```
Env.getCheckpointConfig().setMaxConcurrentCheckpoints(10)
```

- External checkpoints

  You can configure periodic checkpoints to be persisted externally. Externalized checkpoints write their metadata out to persistent storage and are not automatically cleaned up when the job fails. This way, you will have a checkpoint around to resume from if your job fails.

```
Env.getCheckpointConfig().enableExternalizedCheckpoints(ExternalizeCheckpointCleanup.RETAIN_ON_CANCELLATION)
```

6.2.5.3 Savepoint

The checkpoint can be retained in an external media when a job is cancelled. Flink also has another mechanism, savepoint, to restore job data.

Savepoints are a special implementation of checkpoints. The underlying layer actually uses the checkpointing mechanism. Savepoints are triggered by manual commands and the results are persisted to a specified storage path. Savepoints help users save system state data during cluster upgrade and maintenance. This ensures that the system restores to the original computing state when application termination operations such as shutdown O&M or application upgrade are performed. As a result, end-to-end exactly-once semantics can be ensured.

Similar to checkpoints, savepoints allow saving state to external media. If a job fails, it can be restored from an external source. What are the differences between savepoints and checkpoints?

- Triggering and management: Checkpoints are automatically triggered and managed by Flink, while savepoints are manually triggered and managed by users.
- Function: Checkpoints allow fast recovery when tasks encounter exceptions, including network jitter or timeout. On the other hand, savepoints enable scheduled backup and allow you to stop-and-resume jobs, such as modifying code or adjusting concurrency.
- Features: Checkpoints are lightweight and can implement automatic recovery from job failures and are deleted by default after job completion. Savepoints, on the other hand, are persistent and saved in a standard format. They allow code or configuration changes. To resume a job from a savepoint, you need to manually specify a path.

6.2.5.4 State Storage

When checkpointing is enabled, states are persisted along with checkpoints to prevent data loss and ensure data consistency during restoration. How a state is represented internally, and how and where it is persisted upon checkpoints

depends on the chosen state backend.

By default, states are stored in the memory of TaskManager, and checkpoints are stored in the memory of JobManager. Where states and checkpoints are stored depends on the configuration of StateBackend. Flink provides the following out-of-the-box StateBackends:

- **MemoryStateBackend**: memory-based
- **FsStateBackend**: file system–based, which can be a local or an HDFS file system

- **RocksDBStateBackend**: RockDB-based, which can be used as the storage medium If this parameter is not configured, MemoryStateBackend is used by default.

6.2.5.5 MemoryStateBackend

newMemoryStateBackend(intmaxStateSize,booleanasynchronousSnapshots)

MemoryStateBackend: The construction method is to set the maximum StateSize and determine whether to perform asynchronous snapshot. This storage state is stored in the memory of the TaskManager node, that is, the execution node. Because the memory capacity is limited, the default value of **maxStateSize** for a single state is 5 MB. Note that the value of **maxStateSize** is less than or equal to that of **akka.framesize** (10 MB by default). Since the JobManager memory stores checkpoints, the checkpoint size cannot be larger than the memory of JobManager. Recommended scenarios: local testing and jobs that do hold little state, for example, ETL, and JobManager is unlikely to fail, or the failure has little impact. MemoryStateBackend is not recommended in production scenarios.

6.2.5.6 FsStateBackend

FsStateBackend(URIcheckpointDataUri,booleanasynchronousSnapshots)

FsStateBackend: The construction method is to transfer a file path and determine whether to perform asynchronous snapshot. The FsStateBackend also holds state data in the memory of the TaskManager, but unlike MemoryStateBackend, it doesn't have the 5 MB size limit. In terms of the capacity limit, the state size on a single TaskManager cannot exceed the memory size of the TaskManager and the total size cannot exceed the capacity of the configured file system. Recommended scenarios: jobs with large state, such as aggregation at the minute-window level and join, and jobs requiring high-availability setups.

6.2.5.7 RocksDBStateBackend

RocksDBStateBackend(URIcheckpointDataUri,boolean enableIncremental-Checkpointing)

RocksDBStateBackend: RocksDB is a key-value store. Similar to other storage systems for key-value data, the state is first put into memory. When the memory is about to run up, the state is written to disks. Note that RocksDB does not support synchronous Checkpoints. The synchronous snapshot option is not included in the constructor. However, the RocksDBStateBackend is currently the only backend that supports incremental Checkpoints. This suggests that users only write incremental state changes, without having to write all the states each time. The external file systems (local file systems or HDFS) store the checkpoints. The state size of a single TaskManager is limited to the total size of its memory and disk. The maximum size of a key is 2 GB. The total size cannot be larger than the capacity of the configured file system. Recommended scenarios: jobs with very large state, for example, aggregation at the day-window level, jobs requiring high-availability setups, and jobs that do not require high read/write performance.

6.3 Quiz

1. What are the similarities and differences between MapReduce and Spark?

2. How is the exactly-once semantics implemented in Flink and how is the state stored?
3. What are the three time windows of Flink? What are the use cases?

7 Flume's Massive Log Aggregation & Kafka's Distributed Messaging System

7.1 Flume: Massive Log Aggregation

7.1.1 Overview and Architecture

7.1.1.1 Introduction to Flume

Flume is a distributed, reliable, and highly available service used to collect, aggregate, and move large volumes of log data.

It allows users to customize various data senders in the log system, ingests and roughly processes log data in real time, and writes the data to different destinations (such as text, HDFS, and HBase). It also supports cascading (connecting multiple Flumes) and data conflation.

7.1.1.2 Flume Architecture

Agents are the core of Flume. They are the smallest operational unit of Flume. Each agent is a Java Virtual Machine (JVM). It is a comprehensive data collection tool consisting of three core components: sources, channels, and sinks.

- Source: A source is the component that receives data from other systems, sinks of other agents, or even from the source itself. Each type of data is received differently. You can monitor one or more network ports or read data directly from the local file system. Each source connects to at least one channel and copies events to some or all channels. Source can receive data from any source and write the data to one or more channels. A source collects data. It needs to process various types and formats of log data, including Avro, Thrift, and Exec. The following table describes the common source types.

Table 7-1 Source types

Source Type	Description
Exec source	Executes a command or script and uses the output of the execution result as the data source.
Avro source	Provides an Avro-based server, which is bound to a port, and waits for the data sent from the Avro client.

Source Type	Description
Thrift source	Same as Avro, but the transmission protocol is Thrift.
HTTP source	Sends data via the POST request of HTTP.
Syslog source	Collects syslogs.
Spooling directory source	Collects local static files.
JMS source	Obtains data from the message queue.
Kafka source	Obtains data from Kafka.

- Channel: A channel is located between a source and a sink. It caches the events that have been received by an agent but not been written into another agent or HDFS. The channel ensures that the source and sink run securely at different rates. Events are temporarily stored in the channel of each agent and transferred to the next agent or HDFS. Events are deleted from the channel only after they have been successfully stored in a channel or in HDFS of the next agent. Events can be written by sources to one or more channels, and then read by one or more sinks. Common channel types are as follows:
  - Memory channel: Messages are stored in the memory, providing high throughput but not ensuring reliability. This means data may be lost.
  - File channel: Data is persisted; however, the configuration is complex. You need to configure the data directory and checkpoint directory (for each file channel).
  - JDBC channel: A built-in Derby database makes events persistent with high reliability. This channel can replace the file channel that also supports data persistence.
- Sink: A sink is a component that receives events from channels and writes them to the next phase or to their final destination. Sinks continuously poll events in channels, write them to HDFS or other agents in batches, and then batch remove them from channels. In this manner, a source can continuously receive events and write them to channels. The final destinations include HDFS, Logger, Avro, and Thrift. The following table describes the common sink types.

Table 7-2 Sink types

Sink Type	Description
HDFS sink	Writes data to HDFS.
Avro sink	Sends data to Flume of the next hop using the Avro protocol.
Thrift sink	Same as Avro, but the transmission protocol is Thrift.

Sink Type	Description
File roll sink	Saves data to the local file system.
HBase sink	Writes data to HBase.
Kafka sink	Writes data to Kafka.
MorphlineSolr sink	Writes data to Solr.

- Event: The basic unit of data transmission of Flume is an event. A data unit contains an optional message header. Event headers are not used to transmit data. An event ID or universally unique identifier (UUID) is added to the entire event to determine routes or transmit other structural information. The event body is a byte array, which contains the actual transmission load of Flume. If the event body is a text file, it is usually a row of records, which is the basic unit of a transaction.
- Channel processor: used to cache the data sent from the source into the channel.
- Interceptor: a simple plug-in component between a source and a channel. Before a source writes received events to channels, the interceptor can convert or delete the events. Each interceptor processes only the events received by a given source. The interceptor can be customized.
- Channel selector: used to transmit and place data into different channels based on user configurations.
- Sink runner: runs a sink group to drive a sink processor. The sink processor drives sinks to obtain data from channels.
- Sink processor: used to drive sinks to obtain data from channels by configuring policies. Currently, the policies include load balancing, failover, and pass-through.

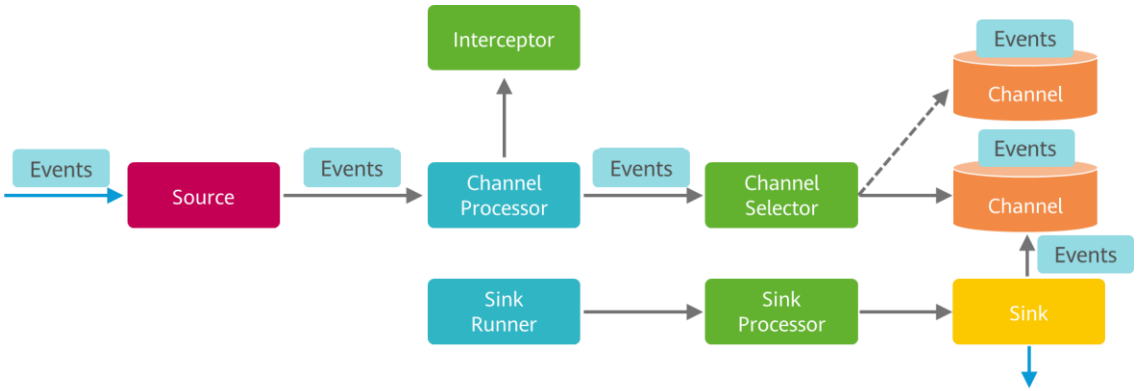


Figure 7-1 Flume architecture

7.1.2 Key Features

7.1.2.1 Log File Collection

Flume collects log files outside a cluster and archives them in HDFS, HBase, or Kafka used for data analysis and cleansing for upper-layer applications. The following figure shows the architecture.

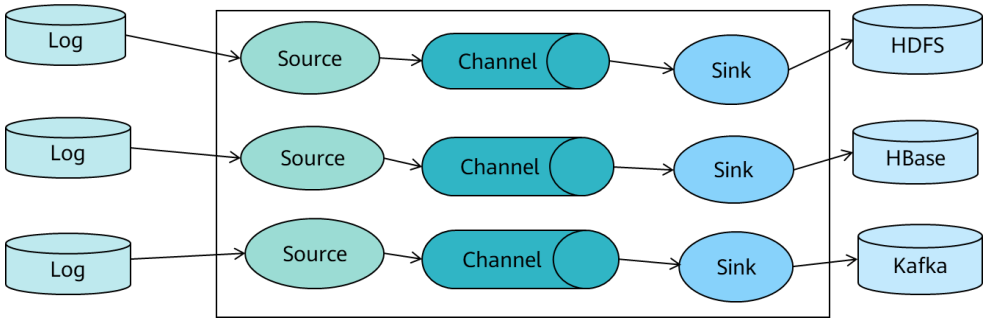


Figure 7-2 Collecting log files

7.1.2.2 Multi-level Cascading and Multi-channel Replication

Flume supports cascading of multiple Flume agents. In this scenario, logs on nodes outside the cluster are collected and aggregated to the cluster through multiple Flume agents. The following figure shows the architecture.

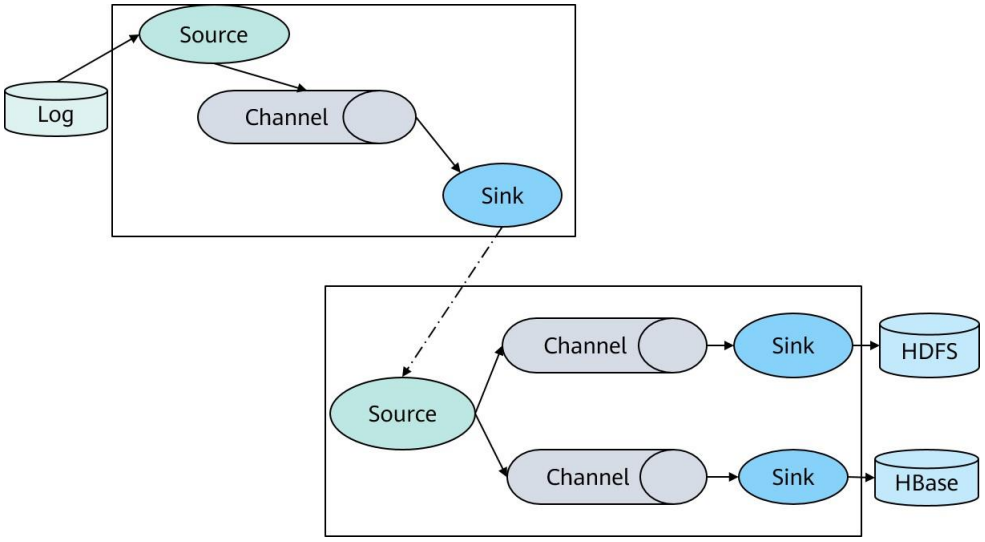


Figure 7-3 Multi-level cascading and multi-channel replication

7.1.2.3 Cascading Message Compression and Encryption

Data transmission between cascaded Flume agents can be compressed and encrypted, improving data transmission efficiency and security. There is no need to encrypt data transmitted from the source to the channel, and then to the sink, for which data is exchanged within a process. The following figure shows the architecture.

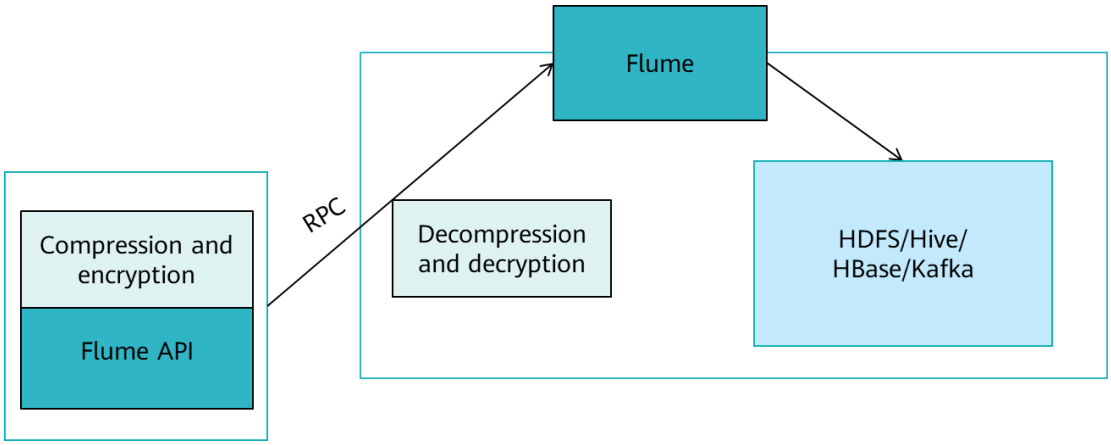


Figure 7-4 Message compression and encryption

7.1.2.4 Transmission Reliability

During data transmission, Flume uses two independent transactions to transfer events from a source to a channel and from a channel to a sink. Once all events in the transaction are transferred to a channel and submitted, the source marks the file as completed. The transaction processes the transfer from a channel to a sink in a similar way. If the event cannot be recorded for some reasons, the transaction will be rolled back. In addition, all events are stored in the channel, where they wait to be transferred again. Flume uses the transaction management mechanism to ensure data completeness and keeps transmission reliable. In addition, if the data is cached in the file channel, data will not be lost when the process or agent is restarted. Data can be automatically switched to another channel for transmission when the next-hop Flume agent is faulty or

data receiving is abnormal during Flume data transmission. The following figure shows the process.

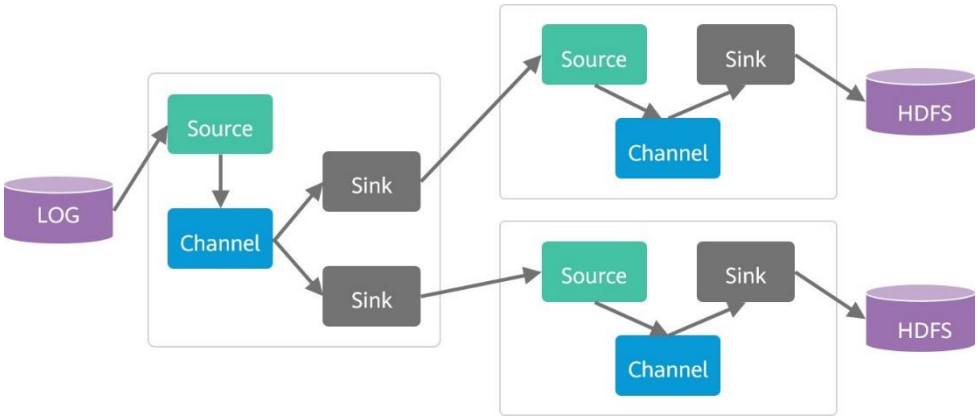


Figure 7-5 Transmission reliability

7.1.2.5 Data Filtering

During data transmission, Flume can filter, clean, and remove unnecessary data. An interceptor is a simple plug-in deployed between a source and a channel. Before the source writes the received events to the corresponding channel, an interceptor can be invoked to convert or delete some events. In a Flume process, any number of interceptors can be added to convert or delete events from a single source. Sources transfer all events of the same transaction to a channel processor, and then the events can be transferred to multiple interceptors, in sequence, until the final event returned by the last interceptor is written to the corresponding channel.

Flume provides multiple types of interceptors, such as timestamp, host, static, and UUID interceptors. If complex data needs to be filtered, users need to develop filter plug-ins based on the particularities of their data. Flume also allows users to invoke third-party filter plug-ins.

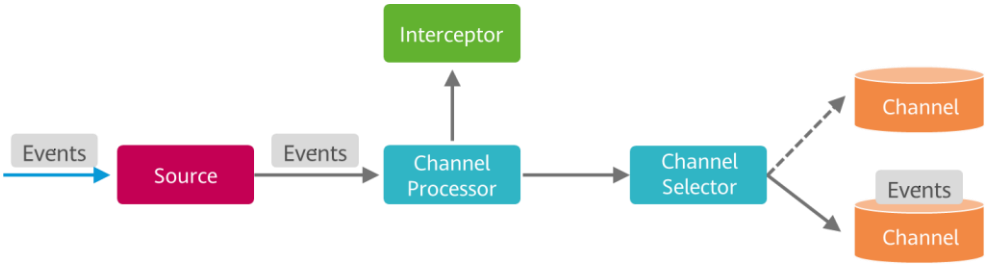


Figure 7-6 Data filtering

7.1.3 Applications

7.1.3.1 Installing and Using Flume

- Download the Flume client: Log in to the MRS Manager and choose **Services** > **Flume** > **Download Client**.
- Install the Flume client: Decompress the client package and install the client.
- Configure the Flume configuration file for the source, channel, and sink.

- Upload the configuration file: Name the configuration file of the Flume agent **properties.properties**, and upload the configuration file.

7.2 Kafka: Distributed Messaging System

7.2.1 Overview

7.2.1.1 Introduction to Kafka

Kafka is a distributed, partitioned, replicated, and ZooKeeper-based messaging system. It supports multi-subscribers and was originally developed by Linkedin. Kafka is commonly used for Web/Nginx logs, access logs, and messaging services. Kafka can process hundreds of thousands of messages per second on ordinary servers. The main application scenarios are log collection systems and messaging systems.

7.2.1.2 Messaging Patterns

Robust message queue is a distributed messaging foundation, where messages are queued asynchronously between client applications and a messaging system. Two types of messaging patterns are available: point to point messaging, and publish-subscribe (pub-sub) messaging. Most messaging patterns are the latter, and Kafka implements a pub-sub messaging pattern.

- Point-to-Point Messaging
  - In a point-to-point messaging system, messages are persisted in a queue. In this case, one or more consumers consume the messages in the queue. However, a message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, the message disappears from that queue. This pattern ensures the data processing sequence even when multiple consumers consume data at the same time. The following figure shows the logical structure.

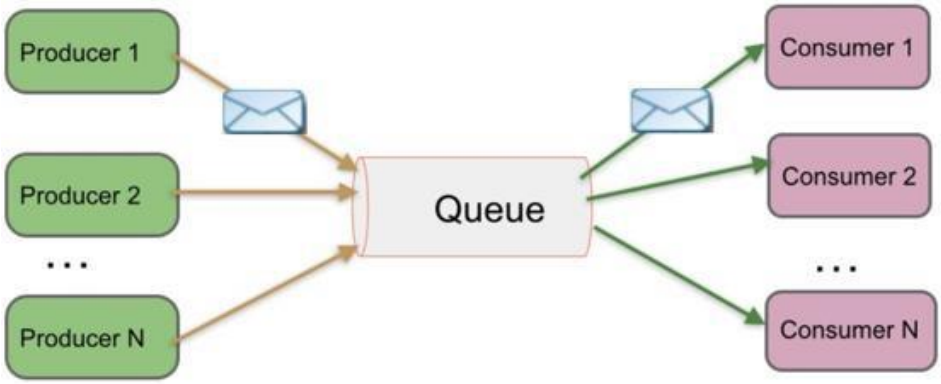


Figure 7-7 Point-to-point messaging

- Publish-Subscribe Messaging
  - In the publish-subscribe messaging system, messages are persisted in a topic. Unlike the point-to-point messaging system, a consumer can subscribe to one or more topics and consume all the messages in those topics. One message can be

consumed by more than one consumer. A consumed message is not deleted immediately. In the publish-subscribe messaging system, message producers are called publishers and message consumers are called subscribers. The following figure shows the logical structure.

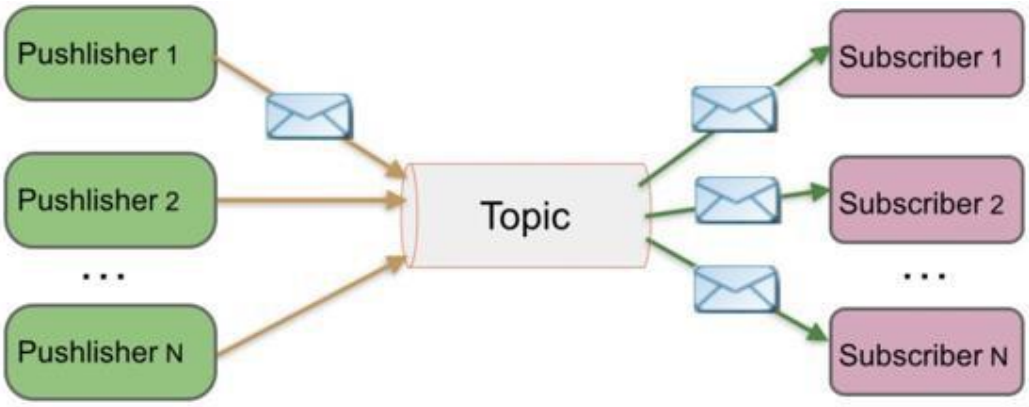


Figure 7-8 Publish-subscribe messaging

7.2.1.3 Kafka Features

- Compressed message set
  - Kafka can send messages in batches and those message sets can be compressed. Producers can compress message sets in GZIP or Snappy format. After compression on the producer side, the sets need to be decompressed on the consumer side. The advantage of compression is that it reduces the amount of data that needs to be transmitted, which eases pressure on the transmission network. In terms of big data processing, the bottleneck is the network instead of the CPU resources (compression and decompression consume some CPU resources). Kafka adds a byte to the message header to describe the compression attribute. The last two digits of the byte are a code used for message compression. If the last two digits are "0", the message was not compressed.
- Message persistence
  - Kafka depends on the file system to store and cache messages. Most people think that a hard disk will always be slow. They tend to be skeptical of what sort of performance that can be provided by a file system-based architecture. In fact, the speed of the hard disk depends on what it is used for. To improve performance, modern operating systems usually use memory as disk caches. All read and write operations on disks pass through the cache. If a program caches a copy of some data in the thread, there is actually another copy in the cache of the operating system. There are effectively two copies of the data. In addition, Kafka based on JVM memory has the following disadvantages: 1. The memory overhead for objects is very high, often twice or even higher than that of the data to be stored. 2. As data increases, garbage collection (GC) becomes slower.
  - The sequential write performance of disks is actually much higher than that of random write. Sequential read/write is greatly optimized by the operating system (read-ahead and write-behind technologies) and is even faster than random memory read/write. Data is not cached in the memory and then flushed to the

hard disk. Instead, data is written directly to the file system logs. Write operations add data to a file in sequence. Read operations read data directly from a file.

- Kafka makes messages persistent so that the stored messages can be used again after the server is restarted. In addition, Kafka supports online or offline processing and integration with other storage and stream processing frameworks.

- Message reliability
  - In a messaging system, the reliability of messages during production and consumption is extremely important. In the actual message transmission process, the following situations may occur: 1. The message fails to be sent. 2. The message is sent multiple times. 3. Each message is sent successfully and only once (exactly-once, ideally).
  - From the perspective of the producer, after a message is sent, the producer waits for a response from the broker (the wait time can be controlled by a parameter). If the message is lost during the process or one of the brokers breaks down, the producer resends the message.
  - From the perspective of the consumer, the broker records an offset value in the partition, which points to the next message to be consumed by the consumer. If the consumer receives a message but cannot process it, the consumer can still find the previous message based on the offset value and process the message again. The consumer has the permission to control the offset value and perform any processing of the messages that are persistently sent to the broker.
- Backup mechanism
  - The backup mechanism improves the reliability and stability of the Kafka cluster. With the backup mechanism, if a node in the Kafka cluster breaks down, the entire cluster is not affected. A cluster whose number of backups is n allows n-1 nodes to fail. Of all the backup nodes, one node serves as the leader node. This node stores the list of other backup nodes and maintains the status synchronization between the backup nodes.
- Lightweight
  - The agency of Kafka is stateless. That is, the agency does not record whether a message is consumed. The consumer or group coordinator maintains the consumption offset management. In addition, the cluster does not need the status information of producers and consumers. Kafka is lightweight, and the implementation of producers and consumer clients is also lightweight.
- High throughput
  - High throughput is the main objective of Kafka design. Kafka messages are continuously added to files. This feature enables Kafka to fully utilize the sequential read/write performance of disks. Sequential read/write does not require the seek time of the disk head. It only requires a small sector rotation time. The speed is much faster than random read/write. Since Linux kernel 2.2, there has been zero-copy calling available. The **sendfile()** function is used to directly transfer data between two file descriptors. It skips the copy of the user buffer and establishes a direct mapping between the disk space and the memory.

Therefore, data is no longer copied to the user-mode buffer. The number of system context switching times is reduced to 2 times, which doubles the performance.

7.2.2 Architecture and Functions

7.2.2.1 Functions of Kafka Roles

- Topic: Kafka abstracts a group of messages into a topic. A topic is a classification of messages. A producer sends messages to a specific topic, and a consumer subscribes to the topic or some portion of the topic for consumption. Each message released to Kafka has a category, which is called a topic or a queue for storing messages. For example, weather can be regarded as a topic (queue) that stores daily temperature information. The following figure shows the structure.

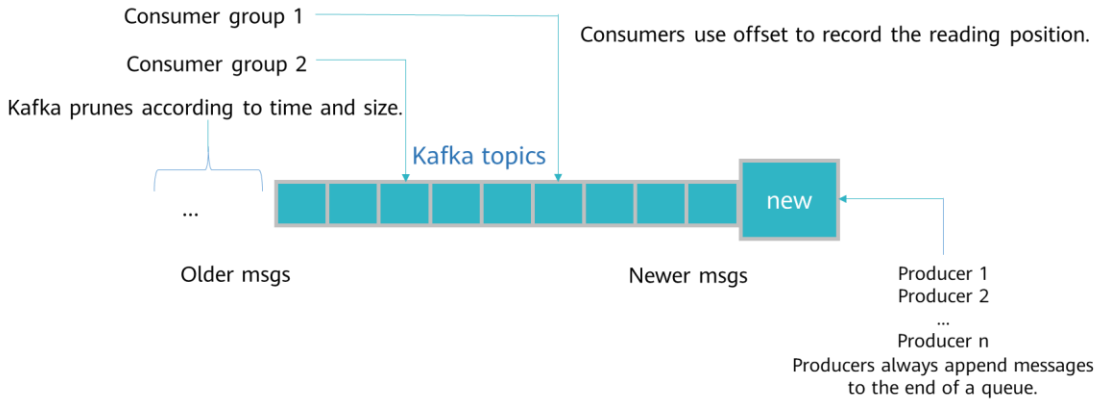


Figure 7-9 Topic

- Partition: To improve the throughput of Kafka, each topic is physically divided into one or more partitions. Each partition is an ordered and immutable sequence of messages. Each partition physically corresponds to a directory for storing all messages and index files of the partition. Partitioning makes it easier for Kafka to process concurrent requests. Theoretically, more partitions mean higher throughput, but it depends on the actual cluster environment and service scenarios. In addition, partitions are the basis for Kafka to ensure that messages are consumed in sequence and are used for load balancing. Kafka can only ensure the order of messages in a partition. It cannot ensure the order of messages across partitions. Each message is appended to the corresponding partition and is written to the disk in sequence, which improves efficiency. This extra efficiency is important to ensuring high throughput for Kafka. Different from traditional messaging systems, Kafka does not delete the consumed messages immediately.

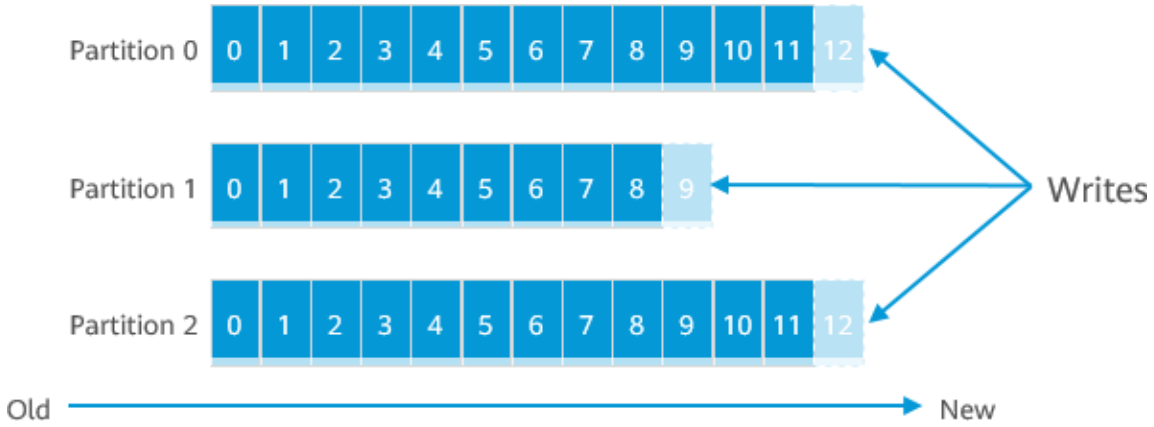


Figure 7-10 Partition

- Offset: The position of each message in a log file is called offset, which is a long integer that uniquely identifies a message. Consumers use offsets, partitions, and topics to track records. The following figure shows the token structure.

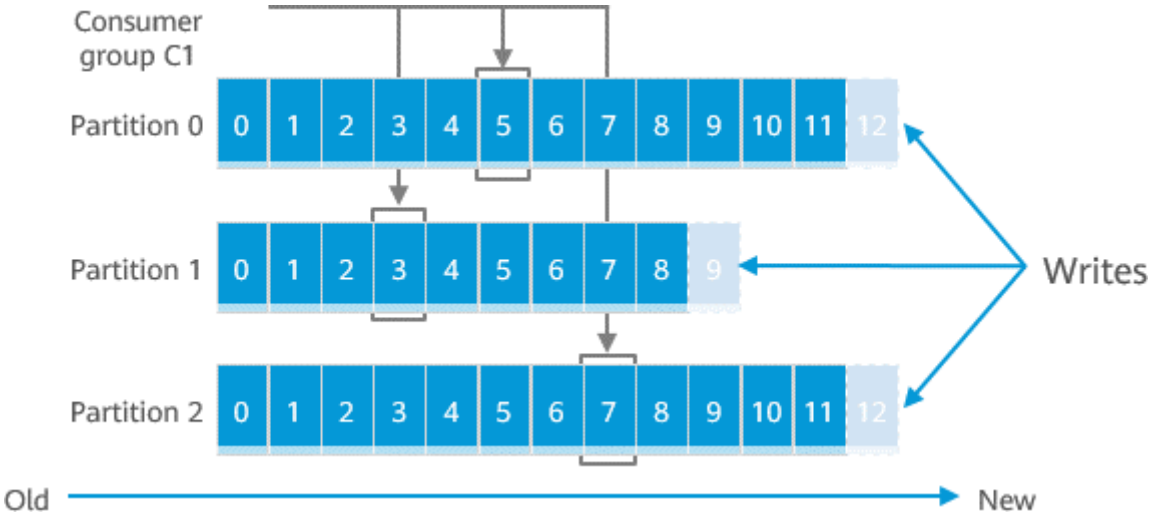


Figure 7-11 Offset

- Offset storage: After a consumer reads the message from the broker, the consumer can commit a transaction, which saves the offset of the read message of the partition in Kafka. When the consumer reads the partition again, the reading starts from the next message. This feature ensures that the same consumer does not consume data repeatedly from Kafka.
- Consumer group: Each consumer belongs to a consumer group. Each message can be consumed by multiple consumer groups but only one consumer in a consumer group. That is, data is shared between groups, but exclusive within a group. The following figure shows the structure.

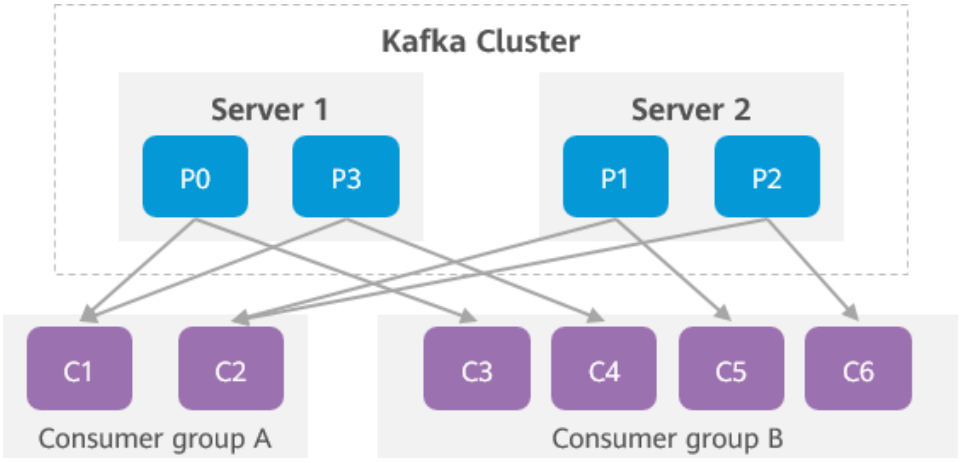


Figure 7-12 Consumer group

Replication: Each partition has one or more replications, which are distributed on different brokers of the cluster to improve availability. The existence of Kafka replications requires data consistency between multiple replications of a partition. Each partition has one server as the leader and may have zero or multiple servers as followers.

The leader processes all read and write operations on the partition. The followers passively replicate data from the leader. If the leader breaks down, the followers automatically elect a new leader. Each server functions as a leader for some partitions and the follower for others, which keeps the load in the Kafka cluster balanced. From the storage perspective, each partition replication is logically abstracted as a log object, meaning partition replications correspond to specific log objects. The number of partitions corresponding to each topic can be configured in the configuration file loaded during Kafka startup or specified during topic creation. The client can change the number of partitions after a topic is created, which is the basis for data storage reliability. The following figure shows the structure of Kafka partition replications.

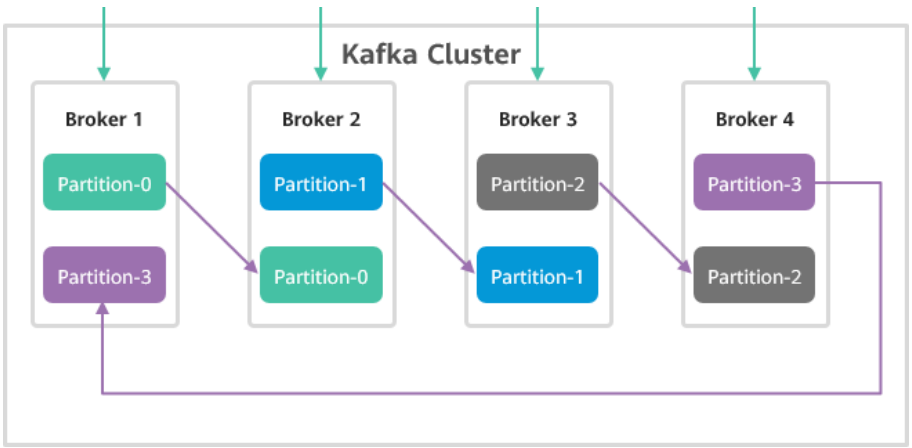


Figure 7-13 Replication

7.2.2.2 Kafka Architecture

A Kafka cluster typically consists of multiple producers (such as page views produced in the web frontend, server logs, system CPUs, and memory), brokers (Kafka supports scale-out. More brokers, higher throughput rate of the cluster), consumers, and a ZooKeeper cluster. The following figure shows the architecture. Kafka uses ZooKeeper to manage

cluster configurations, elect a leader, and rebalance when consumers change. Producers push messages to brokers to publish. Consumers pull messages to brokers to subscribe and consume.

- Record: A message is the basic unit of Kafka communication. Each record is called a message. Each record contains the following attributes:
  - Offset: indicates the unique identifier of a message, which can be used to find a unique message. The corresponding data type is long.
  - Message size: indicates the size of a message. The corresponding data type is int.
  - Data: indicates the content of a message, which can be considered as a byte array.
- Controller: A server in the Kafka cluster, which is used for leader election and various failovers.
- Broker: A Kafka cluster consists of one or more service instances, which are called brokers.
- Producer: releases messages to Kafka brokers.
- Consumer: consumes messages and functions as a Kafka client to read messages from Kafka brokers.

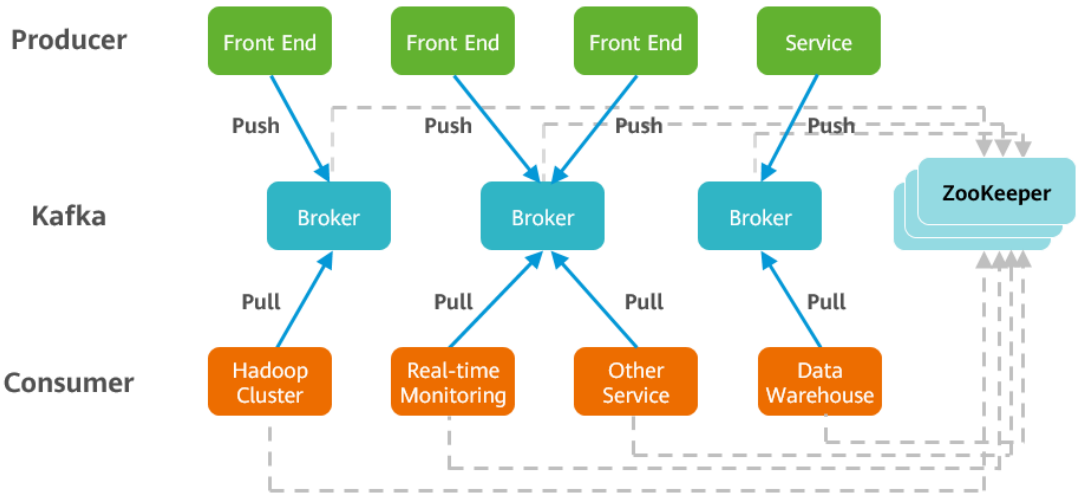


Figure 7-14 Kafka topology

7.2.3 Data Management

7.2.3.1 Data Storage Reliability

To improve fault tolerance, Kafka supports the partition replication policy. That is, the number of partition replicas can be set in the configuration file. Kafka needs to elect a leader for partition replication. The leader is the node responsible for all reads and writes for the given partition, and other replicas are the nodes responsible for data synchronization only. If the leader fails, one of the followers will take over as the new leader. If the synchronized data lags behind too much the leader due to the performance of the follower or network problems, the follower will not be elected as the leader after the leader fails. The server as the leader carries all the request pressure, so from the

overall consideration of the cluster, Kafka will balance leaders across each instance to ensure the overall stable performance. In a Kafka cluster, a node can be a leader and also a follower to another leader. Kafka replications have the following features:

- A replication is based on a partition. Each partition in Kafka has its own primary and secondary replications.
- The primary replication is called a leader, and the secondary replication is called a follower. Followers constantly pull new messages from the leader.
- Consumers and producers read and write data from the leader and do not interact with followers.

In Kafka, data is synchronized between partition replications. A Kafka broker only uses a single thread (ReplicaFetcherThread) to replicate data from the leader of a partition to the follower. Actually, the follower (a follower is equivalent to a consumer) proactively pulls messages from the leader in batches, which greatly improves the throughput.

When a Kafka broker is started, a ReplicaManager is created. ReplicaManager maintains the link connections between ReplicaFetcherThread and other brokers. The leader partitions corresponding to the follower partitions in the broker are distributed on different brokers. These brokers create the same number of ReplicaFetcherThread threads to synchronize the corresponding partition data. In Kafka, every partition follower (acts as a consumer) reads messages from the partition leader. Each time a follower reads messages, it updates the HW status (High Watermark, which indicates the last message successfully replicated to all partition replications). Each time when the broker, where the leader is located, is affected after the partitions of the follower are changed, ReplicaManager creates or destroys the corresponding ReplicaFetcherThread.

A new leader needs to be elected in case of the failure of an existing one. When a new leader is elected, the new leader must have all the messages committed by the old leader. Kafka is not fully synchronous or asynchronous. It is an ISR mechanism:

- The leader maintains a replica list that is basically in sync with the leader. The list is called ISR (in-sync replica). Each partition has an ISR.
- If a follower is too far behind a leader or does not initiate a data replication request within a specified period, the leader removes the follower from ISR.
- The leader commits messages only when all replicas in ISR send ACK messages to the leader.

If all replicas do not work, there are two solutions:

- Wait for a replica in the ISR to return to life and select it as the leader. This can ensure that no data is lost, but may take a long time.
- Select the first replica (not necessarily in the ISR) that returns to life as the leader. Data may be lost, but the unavailability time is relatively short.

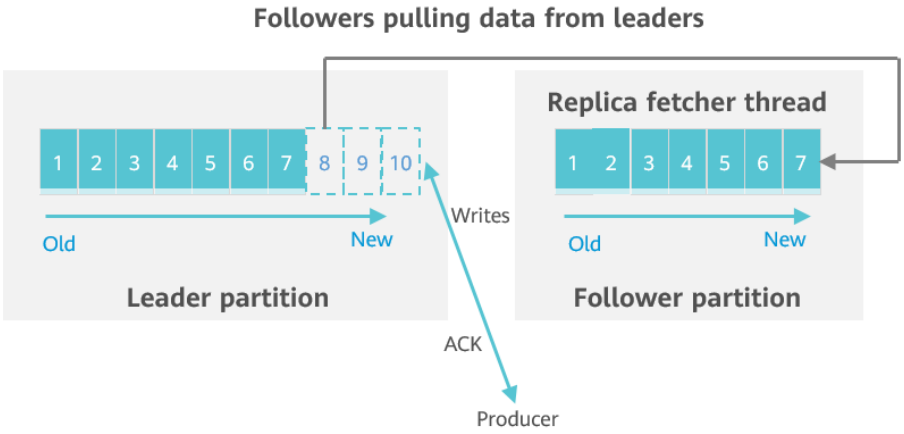


Figure 7-15 Kafka data synchronization

7.2.3.2 Data Transmission Reliability

In a messaging system, the reliability of messages during production and consumption is extremely important. In the actual message transmission process, the following situations may occur:

- At most once: Messages may be lost and will be never redelivered or reprocessed.
- At least once: Messages are not lost but may be redelivered and reprocessed.
- Exactly once: Messages are not lost and are processed only once.

Producers can asynchronously and concurrently send messages to Kafka. After messages are sent, a future response containing the offset value or errors encountered during message sending is returned. The **acks** parameter is very important.

- **acks=0**: indicates that the producer will not wait for any acknowledgment from the server. The record will be immediately added to the socket buffer and considered sent. There are no guarantees that the server has received the record in this case, and the retry configuration will not take effect (as the client will not generally know of any failures). The offset given back for each record will always be set to **–1**. However, setting **acks** to **0** results in the maximum system throughput.
- **acks=1**: indicates that the leader will write the record to its local log but will respond without waiting for full acknowledgement from all followers. In this case, if the leader fails immediately after acknowledging the record but before the followers have replicated it, the record will be lost.
- **acks=all**: indicates that the producer obtains the acknowledgment from the broker when all backup partitions receive messages. This setting ensures the highest reliability but has the lowest efficiency.

In this case, the **acks** parameter is used to ensure the availability of data backups. It is important to ensure the idempotency of data transmission. Idempotent operations produce the same result no matter how many times they are performed. Kafka uses the following operations to ensure idempotence:

- Each batch of messages sent to Kafka will contain a sequence number that the broker will use to deduplicate data.

- The sequence number is made persistent to the replica log. Therefore, if the leader of the partition fails, other brokers take over. The new leader can still determine whether the resent message is duplicate.

The overhead of this mechanism is very low: Each batch of messages has only a few additional fields.

7.2.3.3 Old Data Processing Methods

Kafka stores data permanently to hard disks, but allows users to configure policies to delete or compress data. In Kafka, each partition of a topic is sub-divided into segments, making it easier for periodical clearing or deletion of consumed files to free up space.

For traditional message queues, messages that have been consumed are deleted. However, the Kafka cluster retains all messages regardless of whether they have been consumed. Due to disk restrictions, it is impossible to permanently retain all data (actually unnecessary). Therefore, Kafka needs to process old data. There are two ways to clear old data: delete and compact.

- Delete
  - There are two thresholds for deleting logs: retention time limit and size of all logs in a partition, as described in the following table.

Table 7-3 Deletion thresholds

Parameter	Default Value	Description	Value Range
log.cleanup.policy	delete	Log segments will be deleted when they reach the time limit (beyond the retention time). This can take either the value **delete** or **compact**.	delete or compact
log.retention.hours	168	Maximum period a log segment is kept before it is deleted. Unit: hour	1 – 2147483647
log.retention.bytes	-1	Maximum size of log data in a partition. By default, the value is not restricted. Unit: byte	-1 – 9223372036854775807

- Compact

- Data is compressed and only the data of the last version of each key is retained. In the broker configuration, set **log.cleaner.enable** to **true** to enable the cleaner. By default, the cleaner is disabled. Enable the compression policy by configuring **log.cleanup.policy=compact** in the topic configuration. The following figure shows the compression details.

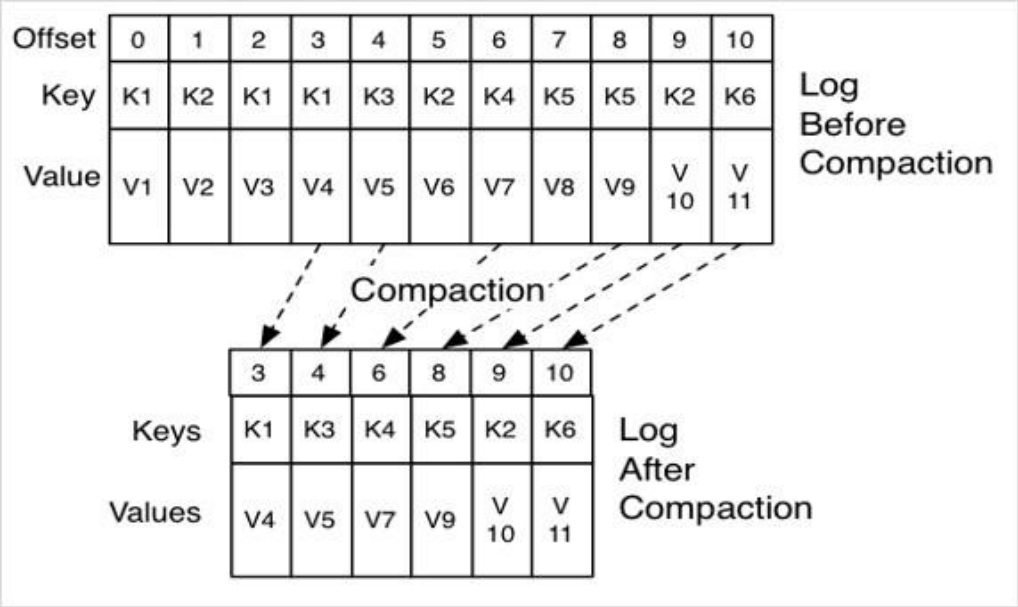


Figure 7-16 Log compression

7.3 Quiz

1. What are the functions of the source, sink, and channel of Flume?
2. What are the functions of ZooKeeper for Kafka?

8 Elasticsearch — Distributed Search Engine

8.1 Overview

In our daily lives, we use search engines to search for movies, books, goods on e-commerce websites, or resumes and positions on recruitment websites. Elasticsearch is often first brought up when we talk about the search function during project development.

In recent years, Elasticsearch has developed rapidly and surpassed its original role as a search engine. It has added the features of data aggregation analysis and visualization. If you need to locate desired content using keywords in millions of documents, Elasticsearch is the best choice.

Elasticsearch is a high-performance Lucene-based full-text search service. It is a distributed RESTful search and data analysis engine and can also be used as a NoSQL database.

- It extends Lucene and provides a query language even richer than that provided by Lucene. The configurable and scalable Elasticsearch optimizes the query performance and provides thorough function management GUIs.
- The prototype environment and production environment can be seamlessly switched. Users can communicate with Elasticsearch in the same way regardless of whether it runs on one node or runs on a cluster containing 300 nodes.
- Elasticsearch supports horizontal scaling. It can process a huge number of events per second, and automatically manage indexes and query distribution in the cluster, facilitating operations.
- Elasticsearch supports multiple data formats, including numerical, text, location data, structured data, and unstructured data.

8.1.1 Elasticsearch Features

Elasticsearch supports multi-condition retrieval, statistics, and report generation for structured and unstructured text. It has a comprehensive monitoring system with a series of key metrics on systems, clusters, and query performance. Elasticsearch helps users focus on service logic implementation. This service applies to scenarios such as log search and analysis, spatiotemporal search, time series retrieval and report generation, and intelligent search. Elasticsearch has the following features:

- High performance: Search results can be obtained immediately, and the inverted index for full-text search is implemented.

- Scalability: Horizontal scaling is supported. Elasticsearch can run on hundreds or thousands of servers. The prototype environment and production environment can be seamlessly switched.
- Relevance: Searches results are sorted based on elements (from word frequency or proximate cause to popularity).
- Reliability: Faults (such as hardware faults and network segmentation) are automatically detected, ensuring cluster (and data) security and availability.

8.1.2 Elasticsearch Application Scenarios

Elasticsearch is a distributed open-source search and analysis engine that is compatible with all types of data, including text, numerical, location data, structured data, and unstructured data. Elasticsearch is known for its simple RESTful APIs, distributed features, high performance, and scalability. Elasticsearch is used in the following scenarios:

- Searching complex types of data. The types include structured data (relational databases), semi-structured data (web pages and XML files), and unstructured data (logs, pictures, and images). Elasticsearch can perform operations such as cleaning, word segmentation, and creation of inverted indexes for all of these data types, and then provide full-text search.
- Searching diverse types of data. For example, there may be too many fields involved, and standard queries cannot meet requirements. Full-text search criteria can include words or phrases.
- Write and read: The written data can be searched in real time.

8.1.3 Elasticsearch Ecosystem

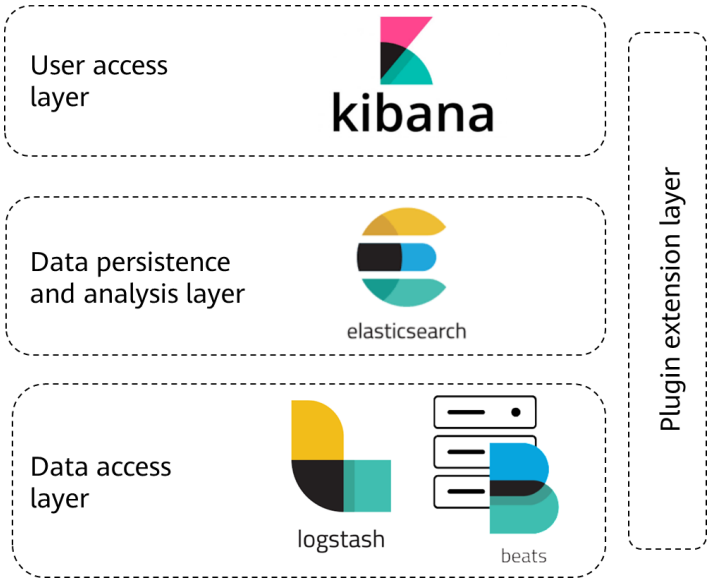


Figure 8-1 ELK structure

ELK/ELKB provides a complete set of solutions. They are open-source software and work together to meet diverse requirements. The plug-in extension layer may include the following components:

Logstash: real-time data transmission pipeline for log processing. It transmits data from the input end to the output end. It provides powerful filters to meet personalized requirements.

Kibana: open-source analysis and visualization platform for data on Elasticsearch. Users can obtain results required for upper-layer analysis and visualization. Developers or O&M personnel can easily perform advanced data analysis and view data in charts, tables, and maps.

Beats: platform dedicated to data transmission. It can seamlessly transmit data to Logstash or Elasticsearch. Installed with the lightweight agent mechanism, it is similar to the Ambari or CDH Manager used during Hadoop cluster installation. Beats sends data from hundreds or thousands of computers to Logstash or Elasticsearch.

Elasticsearch-hadoop: integrated with Hadoop and Elasticsearch and subproject officially maintained by Elasticsearch. Data is input and output between Hadoop and Elasticsearch. With the parallel computing of MapReduce, real-time search is available for HDFS data.

Elasticsearch-sql: uses SQL statements to perform operations on Elasticsearch, instead of writing complex JSON queries. Currently, Elasticsearch-sql has two versions: one is the open-source nlpchina/Elasticsearch-sql plugin promoted in China many years ago, and the other is the Elasticsearch-sql officially supported after Elasticsearch 6.3.0 was released in June 2018.

Elasticsearch-head: client tool for Elasticsearch. It is a GUI-based cluster operation and management tool that is used to perform simple operations on clusters, and is a frontend project based on Node.js.

Bigdesk: cluster monitoring tool for Elasticsearch. Users can use it to view the status of the Elasticsearch cluster, such as the CPU and memory usages, index data, search status, and number of HTTP connections.

8.2 Elasticsearch Architecture

8.2.1 System Architecture

Elasticsearch uses a distributed cluster architecture. Multiple Elasticsearch instances form a cluster, and each cluster has a master node. Elasticsearch is decentralized. Therefore, the master node is dynamically elected and there is no single point of failure (SPOF). In the same subnet, you only need to set the same cluster name on each node. Then Elasticsearch automatically combines the nodes with the same cluster name into a cluster. The communication between nodes as well as data distribution and balancing between nodes are automatically managed by Elasticsearch. Externally, Elasticsearch is an entire system.

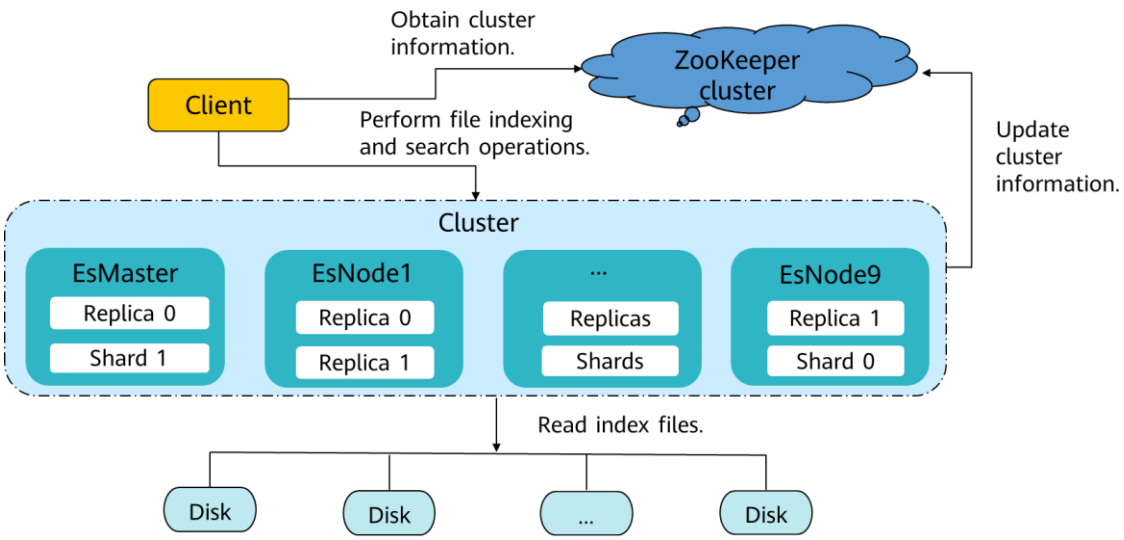


Figure 8-2 Elasticsearch system architecture

As shown in the preceding figure, the Elasticsearch cluster consists of the EsMaster and EsNode processes. The roles and their functions are as follows:

- Client: The client communicates with EsMaster and EsNode instance processes in the Elasticsearch cluster through HTTP or HTTPS, and performs distributed indexing and search operations.
- Cluster: Each cluster contains multiple nodes, one of which is the master node. The master node can be elected. A master node or slave node is distinguished inside a cluster.
- EsMaster: EsMaster is the master node and stores Elasticsearch metadata. It temporarily manages cluster-level changes, such as creating or deleting indexes, and adding or removing nodes. A master node does not involve in document-level change or search. When the traffic increases, the master node does not affect the cluster performance.
- EsNode: An EsNode is an Elasticsearch node. Each node is an Elasticsearch instance that stores the index data of Elasticsearch. Shard: an index shard. Elasticsearch can split a complete index into multiple shards and distribute them on different nodes.
- Replica: an index replica. Multiple index replicas can be set for Elasticsearch. Replicas are used to improve the fault tolerance of the system. When a shard of a node is damaged or lost, the shard can be restored from the replicas. The other is to improve the query efficiency of Elasticsearch. Elasticsearch automatically balances the load of search requests.
- ZooKeeper: mandatory in Elasticsearch. It provides functions such as storage of security authentication information.

8.2.2 Elasticsearch Internal Architecture

Elasticsearch provides various access APIs through RESTful APIs or other languages (such as Java), uses the cluster discovery mechanism, and supports script languages and various plug-ins. The underlying layer is based on Lucene and maintains absolute independence

of Lucene. Index storage is implemented through local files, shared files, and HDFS. The following figure shows the internal architecture of Elasticsearch.

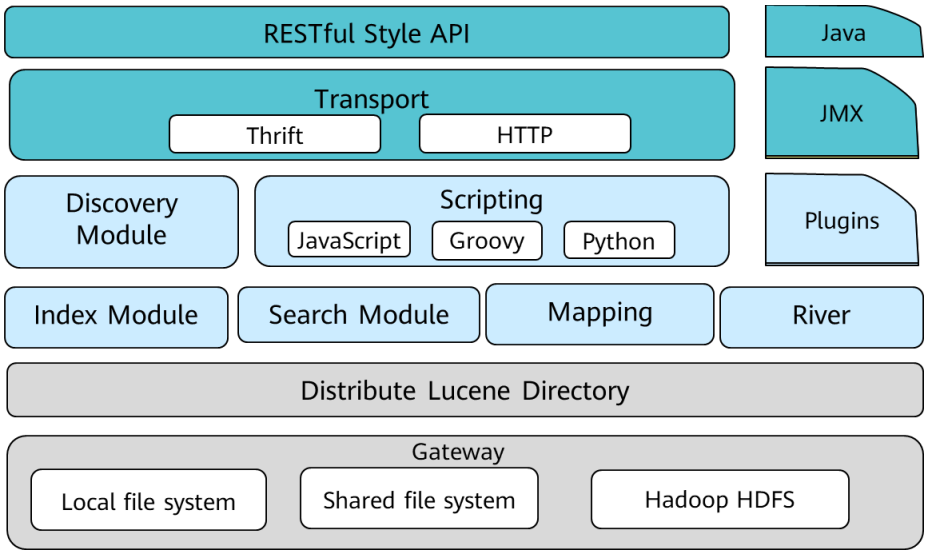


Figure 8-3 Elasticsearch internal architecture

The functions of the modules in the preceding figure are as follows:

- Gateway: Mode for storing Elasticsearch index snapshots. By default, Elasticsearch stores indexes in the memory and only makes them persistent on the local hard disk when the memory is full. The gateway stores index snapshots. When an Elasticsearch cluster is disabled and restarted, the cluster reads the index backup data from the gateway. Elasticsearch supports multiple types of gateways, including the default local file system, distributed file system, Hadoop HDFS, and Amazon S3.
- Distributed Lucene Directory layer: The upper layer of the gateway is a distributed framework for Lucene. Lucene is used for search, but it is a single-node search engine, such as the Elasticsearch distributed search engine system. Although Lucene is used at the bottom layer, Lucene needs to be run on each node for indexing, query, and update, so a distributed running framework is required to meet service requirements.

Four modules of Elasticsearch:

- The index module is used to create indexes for data, that is, to create some inverted indexes.
- The search module is used to query and search for data.
- The mapping module is used to map and parse data. Each field of data is mapped and parsed based on the established table structure. If no table structure is established, Elasticsearch infers the data structure based on the data type, generates a mapping, and then parses data based on the mapping.
- River represents a data source of Elasticsearch and is also a method for synchronizing data to Elasticsearch in other storage modes (such as databases, River, couchDB, RabbitMQ, Twitter, and Wikipedia). It is an Elasticsearch service that exists as a plugin, which reads data from the River and indexes it to Elasticsearch.

Discovery layer: This module is responsible for automatic node discovery and master node election in a cluster. Nodes communicate with each other in P2P mode, eliminating single points of failure. In Elasticsearch, the master node maintains the global status of the cluster. For example, if a node is added to or removed from the cluster, the master node reallocates shards.

Script layer: Elasticsearch query supports multiple script languages, including MVEL, JS, and Python.

Transport layer: Interaction mode between an Elasticsearch internal node or cluster and the client. By default, internal nodes use the TCP protocol for interaction. In addition, transmission protocols (integrated using plugins) as the HTTP (JSON format), Thrift, Servlet, Memcached, and ZeroMQ are supported.

RESTful interface layer: The top layer is the access interface exposed by Elasticsearch. The recommended solution is a RESTful interface, which directly sends HTTP requests to facilitate the use of Nginx as a proxy and distribution. In addition, permission management may be performed in the future. It is easy to perform such management through HTTP.

8.2.3 Elasticsearch Core Concepts

- Index
  - An index is a logical namespace which maps to one or more shards. Apache Lucene is used to read and write data in the index. An index is similar to a relational database instance. An Elasticsearch instance can contain multiple indexes.
- Document
  - A document is a basic unit of information that can be indexed. This document refers to JSON data at the top-level structure or obtained by serializing the root object. A document is equivalent to a row in a database. A document consists of fields. Each field contains a field name and one or more field values. In this case, the field is called a multi-value field. That is, the document contains multiple fields with the same name. There can be different sets of fields between documents, and there is no fixed schema or mandatory structure.
- Type
  - Each document in Elasticsearch has a corresponding type. Types allow users to store multiple document types in one index and provide different mappings for different document types. If compared with the SQL domain, it is equivalent to a table in the database.
- Mapping
  - A mapping is used to restrict the type of a field and can be automatically created based on data. A mapping is similar to a schema in a database.

8.3 Key Features

8.3.1 Elasticsearch Cache Mechanism

Elasticsearch caches are classified into Query Cache, Field Data Cache, and Request Cache.

- Query Cache: node-level cache, which caches the filter execution results contained in a query.
- Field Data Cache: caches the data structure of the word segmentation field during query.
- Request Cache: shard-level cache, which caches local result sets at the shard level.

8.3.2 Elasticsearch Inverted Index

The traditional search method (forward indexing) searches for specific information that meets the search criteria based on keywords, that is, searches for values based on keys. However, Elasticsearch (Lucene) uses inverted indexes to search for keys based on values. In full-text search, the value is the keyword to be searched for. The place where all keywords are stored is called a dictionary. The key is the document number list, through which documents with the searched keyword (documents with the value) can be filtered out, as shown in the following figure.

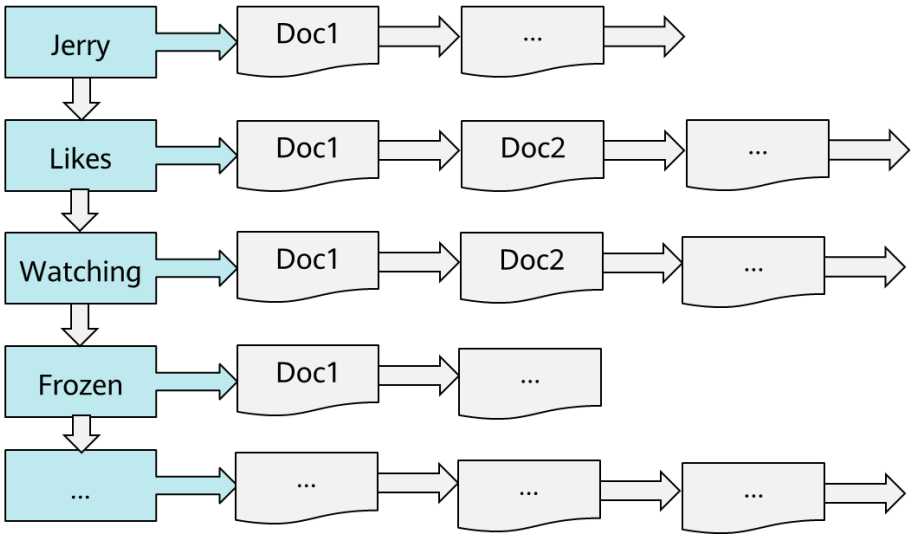


Figure 8-4 Keyword search process

The reverse index search is to search for the corresponding document number through the keyword, and then search for the document through the document number. This is similar to looking up a dictionary or searching for the content of the specified page through the book catalog.

8.3.3 Elasticsearch Routing Algorithm

When a document is indexed, it is stored in a primary shard. How does Elasticsearch determine which shard a document should be stored in? When we create a document, how does it decide whether it should be stored in shard 1 or shard 2? First of all, this is

not random, or we would not know where to find the documents in the future when we want to get them.

Elasticsearch provides two routing algorithms:

- Default route: shard=hash (routing)%number_of_primary_shards. This routing policy is limited by the number of shards. During capacity expansion, the number of shards needs to be multiplied (Elasticsearch 6.x). In addition, when creating an index, you need to specify the capacity to be expanded in the future. Note that Elasticsearch 5.x does not support capacity expansion, but Elasticsearch 7.x supports free expansion.
- Custom route: In this routing mode, the routing can be specified to determine the shard to which a document is written, or search for a specified shard.

8.3.4 Elasticsearch Balancing Algorithm

Elasticsearch automatically balances shards among available nodes, including adding new nodes and taking existing nodes offline. The customer wants to elevate replica shards to restore the lost primary shards as soon as possible and wants to ensure that resources are balanced in the entire cluster to avoid hotspots. For the self-balancing strategy, Elasticsearch provides the following algorithms:

- weight_index(node,index)=indexBalance*(node.numShards(index)
  -avgShardsPerNode(index))
- weight_node(node,index)=shardBalance*(node.numShards()-avgShardsPerNode)
- weight(node,index)=weight_index(node,index)+weight_node(node,index)

8.3.5 Elasticsearch Capacity Expansion

As the service volume increases, Elasticsearch has a large number of tasks, and its storage resources and computing capabilities cannot meet service requirements. In this case, you need to expand the capacity of Elasticsearch. In terms of operations, capacity expansion scenarios can be classified into the following types:

- Instance expansion: adding more Elasticsearch service instances.
- Adding nodes: adding more servers to the cluster.

In application, if Elasticsearch is affected by service volume changes or resource restrictions, you may need to expand or reduce the cluster capacity. Both capacity expansion and reduction are classified as horizontal and vertical adjustment. Horizontal adjustment is when you increase or decrease the number of Elasticsearch service instances. Vertical adjustment is generally when the server performance is scaled up, for example, using better CPUs or faster and larger hard disks. For distributed services such as Elasticsearch, capacity adjustment refers to horizontal adjustment by default. Capacity expansion and reduction are for horizontal adjustment and do not involve vertical adjustment.

8.3.6 Elasticsearch Capacity Reduction

If some Elasticsearch servers need to be brought offline or the number of used Elasticsearch servers needs to be adjusted due to insufficient server resources, you need to reduce the capacity of Elasticsearch. Compared with capacity expansion scenarios, capacity reduction scenarios can be classified into the following types:

- Reducing the number of Elasticsearch service instances
- Removing Elasticsearch servers from the cluster

Precautions for capacity reduction: Ensure that the replica of the shard on the instance to be deleted exists on other instances. Ensure that the data on the instance to be deleted has been migrated to other nodes.

8.3.7 Elasticsearch Indexing HBase Data

When HBase data is written to HDFS, Elasticsearch creates the corresponding HBase index data. The index ID is mapped to the rowkey of the HBase data, which ensures the unique mapping between each index data record and HBase data and implements full-text search of the HBase data. For data that already exists in HBase, an MR task is submitted to read all data in HBase, and then indexes are created in Elasticsearch. The following figure shows the indexing process.

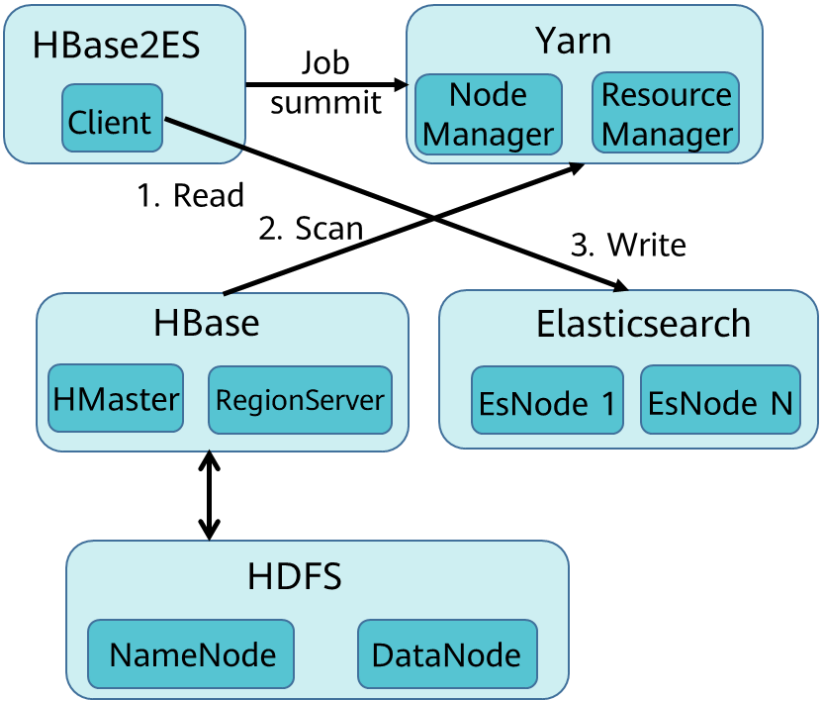


Figure 8-5 Process for Elasticsearch to index HBase data

8.3.8 Elasticsearch Multi-instance Deployment on a Node

Multiple Elasticsearch instances can be deployed on one node, and are differentiated from each other based on the IP address and port number. This method increases the usages of the single-node CPU, memory, and disk, and also improves the indexing and search capability of Elasticsearch.

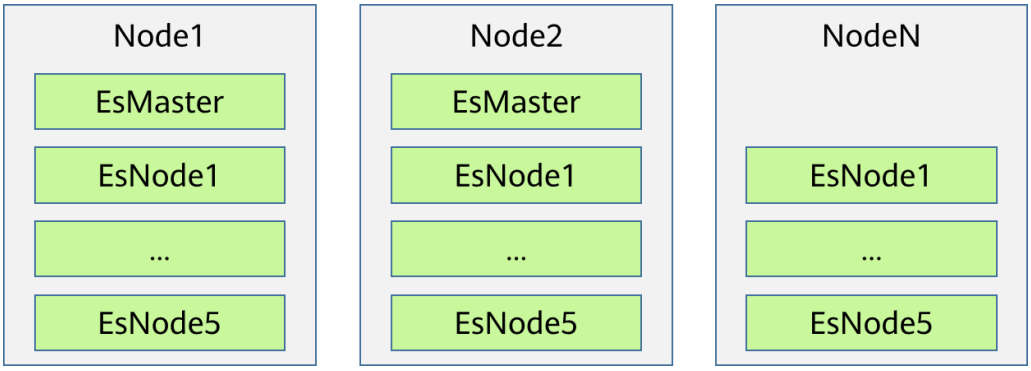


Figure 8-6 Elasticsearch multi-instance deployment on a node

8.3.9 Elasticsearch Cross-Node Replica Allocation Policy

When multiple instances are deployed on a single node with multiple replicas, if the replicas can only be allocated across instances, a single point of failure may occur. To solve this problem, set parameter **cluster.routing.allocation.same_shard.host** to **true**.

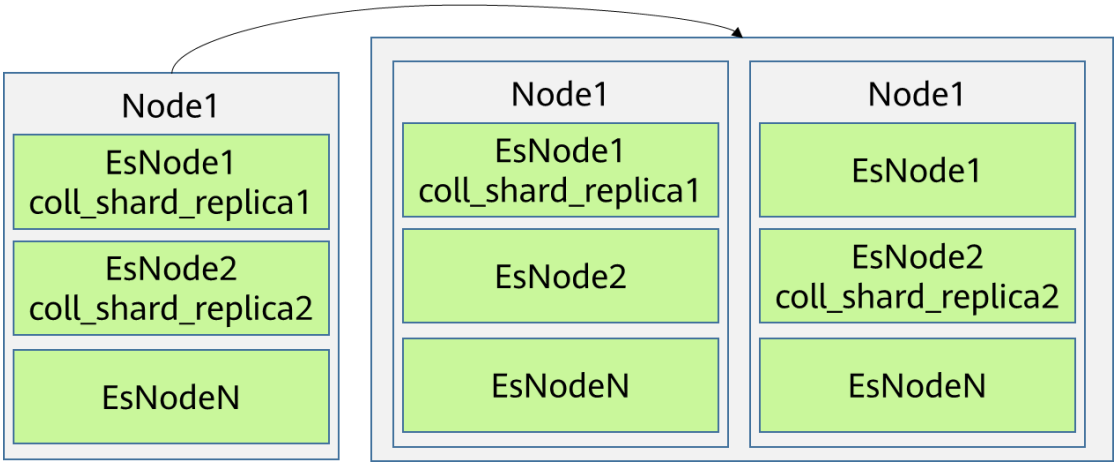


Figure 8-7 Elasticsearch cross-node replica allocation policy

8.4 Quiz

1. How does Elasticsearch implement master election?
2. Describe the data write process of Elasticsearch.

9 Huawei Big Data Platform MRS

9.1 Huawei Big Data Platform MRS

Huawei Cloud big data services are based on MRS, a one-stop big data platform. MRS can interconnect with operations platform DAYU and data visualization services. Enterprises can build a unified big data platform for data access, data storage, data analysis, and value mining to help customers easily resolve difficulties in data channel cloudification, big data job development and scheduling, and data display. Thereby, customers are free from complex big data platform construction and professional big data optimization and maintenance, and they can focus on industry applications and use one piece of data in multiple service scenarios.

Trends in the evolution of data analysis technologies: **data warehouse - > data lake - > lakehouse**

- **What is a data warehouse?**
  - The large-scale application of databases has facilitated the exponential growth of data. Online analytical processing (OLAP) is desired more than ever to explore the relationship between data and unveil the untapped data value. However, it is difficult to share data between different databases, and data integration and analysis also face great challenges.
  - To overcome these challenges for enterprises, Bill Inmon, proposed the idea of data warehousing in 1990. The data warehouse runs on a unique storage architecture to perform OLAP on a large amount of the OLTP data accumulated over years. In this way, enterprises can obtain valuable information from massive data quickly and effectively to make informed decisions. Thanks to data warehouses, the information industry has evolved from operational systems based on relational databases to decision support systems.
  - Unlike a database, a data warehouse has the following features:
  - A data warehouse uses themes. It is built to support various services, with data coming from scattered operational data. Therefore, the required data needs to be extracted from multiple heterogeneous data sources, processed and integrated, and reorganized by theme.
  - A data warehouse mainly supports enterprise decision analysis and operations involved are mainly data queries. Therefore, it improves the query speed and cuts the total cost of ownership (TCO) by optimizing table structures and storage modes.

Table 9-1 Differences between data warehouses and databases

Dimension	Data Warehouse	Database
Application scenarios	OLAP	OLTP
Data source	Multiple	Single
Data normalization	Denormalized schemas	Highly normalized static schemas
Data access	Optimized read operations	Optimized write operations

- **What is a data lake?**
  - Data is an important asset for enterprises. As production and operations data piles up, enterprises hope to save the data for effective management and centralized governance and explore data values.
  - The data lake provides a good answer to these requirements. It is a large data warehouse that centrally stores structured and unstructured data. It can store raw data of multiple data sources and types, meaning that data can be accessed, processed, analyzed, and transmitted without being structured first. The data lake helps enterprises quickly complete federated analysis of heterogeneous data sources and explore data value.
  - A data lake is in essence a solution that consists of a data storage architecture and data processing tools.
  - The storage architecture must be scalable and reliable enough to store massive data of any type (structured, semi-structured, unstructured data).
  - Data processing tools are classified into two types:
  - The first type migrates data into the lake, including defining sources, formulating synchronization policies, moving data, and compiling catalogs.
  - The second type then uses that data, including analyzing, mining, and using it. The data lake must be equipped with wide-ranging capabilities, such as comprehensive data and data lifecycle management, diversified data analytics, and secure data acquisition and release. These data governance tools help guarantee data quality, which can be compromised by a lack of metadata and turn the data lake into a data swamp.
  - Now with big data and AI, lake data is even more valuable and plays new roles. It represents more enterprise capabilities. For example, the data lake can centralize data management, helping enterprises build more optimized operation models. It also provides other enterprise capabilities such as prediction analysis and recommendation models. These models can stimulate further growth.
  - Just like any other warehouse and lake, one stores goods, or data, from one source while the other stores water, or data, from many sources.

Table 9-2 Differences between data lakes and data warehouses

Dimension	Data Lake	Data Warehouse
Application scenarios	Exploratory analytics (machine learning, data discovery, profiling, prediction)	Data analytics (based on historical structured data)
Cost	Low initial cost and high subsequent cost	High initial cost and low subsequent cost
Data quality	Massive raw data to be cleaned and normalized before use	High quality data that can be used as the basis of facts
Target user	Data scientists and data developers	Business analysts

- **What is a lakehouse?**
  - Although the application scenarios and architectures of a data warehouse and a data lake are different, they can cooperate to resolve problems. A data warehouse stores structured data and is ideal for quick BI and decision-making support, while a data lake stores data in any format and can generate larger value by mining data. Therefore, their convergence can bring more benefits to enterprises in some scenarios.
  - A lakehouse, the convergence of a data warehouse and a data lake, aims to enable data mobility and streamline construction. The key of the lakehouse architecture is to enable the free flow of data/metadata between the data warehouse and the data lake. The explicit-value data in the lake can flow to or even be directly used by the warehouse. The implicit-value data in the warehouse can also flow to the lake for long-term storage at a low cost and for future data mining.

9.1.2 Huawei Cloud Services

Huawei Cloud is Huawei's signature cloud service brand. It is a culmination of Huawei's 30-plus years of expertise in ICT infrastructure products and solutions. Huawei Cloud is committed to providing stable, secure, and reliable cloud services that help organizations of all sizes grow in an intelligent world. To complement an already impressive list of offerings, Huawei Cloud is pursuing a vision of inclusive AI, a vision of AI that is affordable, effective, and reliable for everyone. As a foundation, Huawei Cloud provides a powerful computing platform and an easy-to-use development platform for Huawei's full-stack all-scenario AI strategy. Huawei aims to build an open, cooperative, and win-win cloud ecosystem and helps partners quickly integrate into that local ecosystem. Huawei Cloud adheres to business boundaries, respects data sovereignty, does not monetize customer data, and works with partners for joint innovation to continuously create value for customers and partners. By the end of 2019, Huawei Cloud has launched

200+ cloud services and 190+ solutions, serving many well-known enterprises around the world.

9.1.3 Huawei Cloud MRS

Big data is a huge challenge facing the Internet era as data grows explosively both in volume and diversity. Conventional data processing techniques, such as single-node storage and relational databases, are unable to keep up. To meet this challenge, the Apache Software Foundation (ASF) has launched an open-source Hadoop big data processing solution. Hadoop is an open-source distributed computing platform that can fully utilize compute and storage capabilities of clusters to process massive amounts of data. However, if enterprises deploy Hadoop by themselves, they will encounter many problems, such as high costs, long deployment periods, difficult maintenance, and inflexible use.

To solve these problems, Huawei Cloud provides MRS for managing the Hadoop system. With MRS, you can deploy the Hadoop cluster in just one click. MRS provides enterprise-level big data clusters on the cloud. Tenants have full control over clusters and can easily run big data components such as Hadoop, Spark, HBase, Kafka, and Storm. MRS is fully compatible with open-source APIs, and incorporates the advantages of Huawei Cloud computing and storage and big data industry experience to provide customers with a full-stack big data platform featuring high performance, low cost, flexibility, and ease-of-use. In addition, the platform can be customized based on service requirements to help enterprises quickly build a massive data processing system and discover new value points and business opportunities by analyzing and mining massive amounts of data in real time or at a later time.

9.1.4 MRS Highlights

MRS has the following advantages:

- High Performance
  - MRS supports self-developed CarbonData storage technology. CarbonData is a high-performance big data storage solution. It allows one piece of data to be used in multiple scenarios and supports features such as multi-level indexing, dictionary encoding, pre-aggregation, dynamic partitioning, and quasi-real-time data query. This improves I/O scanning and computing performance and returns analysis results of tens of billions of data records within seconds. In addition, MRS supports the self-developed enhanced Scheduler Superior, which breaks the scale bottleneck of a single cluster and is capable of scheduling over 10,000 nodes in a cluster.
- Low cost
  - Based on diversified cloud infrastructure, MRS provides various computing and storage choices and supports storage-compute decoupling, delivering cost-effective mass data storage solutions. MRS supports auto scaling to address peak and off-peak service loads, releasing idle resources on the big data platform for customers. MRS clusters can be created and scaled out when you need them, and can be terminated or scaled in after you use them, minimizing cost.
- Strong security

    - MRS delivers enterprise-level big data multi-tenant permissions management and security management to support table-based and column-based access control and data encryption.
- Easy O&M
    - MRS provides a visualized big data cluster management platform, improving O&M efficiency. MRS supports rolling patch upgrade and provides visualized patch release information and one-click patch installation without manual intervention, ensuring long-term stability of user clusters.
- High reliability
    - The proven large-scale reliability and long-term stability of MRS meet enterprise-level high reliability requirements. In addition, MRS supports automatic data backup across AZs and regions, as well as automatic anti-affinity. It allows VMs to be distributed on different physical machines.

9.1.5 What are the advantages of MRS compared with self-built Hadoop?

- **MRS supports one-click cluster creation, deletion, and scaling, and allows users to access the MRS cluster management system using EIPs.**
- **MRS supports auto scaling, which is more cost-effective than the self-built Hadoop cluster.**
- **MRS supports storage-compute decoupling, greatly improving the resource utilization of big data clusters.**
- **MRS supports Huawei-developed CarbonData and Superior Scheduler, delivering better performance.**
- **MRS optimizes software and hardware based on Kunpeng processors to fully release hardware computing power and achieve cost-effectiveness.**
- **MRS supports multiple isolation modes and multi-tenant permission management of enterprise-level big data, ensuring higher security.**
- **MRS implements HA for all management nodes and supports comprehensive reliability mechanism, making the system more reliable.**
- **MRS provides a big data cluster management UI in a unified manner, making O&M easier.**
- **MRS has an open ecosystem and supports seamless interconnection with peripheral services, allowing you to quickly build a unified big data platform.**

9.1.6 MRS Architecture

One lake per enterprise: large, fast, convergent, and reliable

- Large: Up to 20,000 nodes per cluster
- Fast: All data kept within the lake, faster data retrieval and analysis.
- Convergent: Full convergence of batch, streaming, and interactive data analysis, unified resource scheduling, over 90% of resource utilization.
- Reliable: Hitless upgrade for the latest technology with zero downtime

Real-time, incremental data updates, offline and real-time data warehouses over the same architecture

Real-time data import and analysis are available.

One copy of data can be imported to the database in real time and analyzed from multiple dimensions.

The offline data warehouse can be seamlessly upgraded to a real-time one, allowing for converged batch and stream analysis.

Decoupled storage and compute, EC ratio as low as 1.2, over 20% TCO reduction

Integrated data lake and warehouse

The in-lake interactive engine outperforms same-class products by over 30%. You can generate BI reports using the data in the lake through a self-service graphical interface.

Convergence of batch, streaming, and interactive data analysis via a unified SQL interface.

Collaborative computing across MRS and GaussDB(DWS), no need to move data around.

9.1.7 MRS Application Scenarios

Huawei Cloud FusionInsight MRS cloud-native data lake offers a lakehouse architecture that is capable of continuous evolution and can be used to build three different types of data lakes: offline, real-time, and logical. It applies to real-time analysis, offline analysis, interactive query, real-time retrieval, multi-modal analysis, data warehousing, data ingestion, and data governance. It enables enterprises to use their data in an efficient and simple way, achieve "one lake for one enterprise" and "one lake for one city", gain accurate insight into business, and monetize data more quickly.

9.1.8 MRS in Hybrid Cloud: Data Base of the FusionInsight Intelligent Data Lake

Huawei Cloud FusionInsight intelligent data lake has provided enterprise-level data lake solutions for 3000+ government, finance, carrier, and large enterprise customers in 60+ countries and regions, for example, government data governance and all-in-one network office, real-time financial risk control, carrier-BOM three-domain convergence, and smart campus, smart urban rail, and smart airport for large enterprises. Huawei Cloud FusionInsight includes cloud services such as MRS, GaussDB(DWS), DataArts Studio, and GES. It supports massive data analysis scenarios such as real-time analysis, offline analysis, interactive query, real-time retrieval, multi-modal analysis, data warehouse mart, data access governance, and graph computing of full data of government and enterprise customers, accelerating data value release and helping government and enterprise customers achieve one lake for one enterprise and one lake for one city.

MRS cloud-native data lake: uses one architecture to build logical, real-time, and offline data lakes and therefore provide government and enterprise customers with integrated lakehouse and cloud-native big data solutions. MRS contains components such as Spark, ClickHouse, HetuEngine, Hive, HBase, and Hudi to support real-time analysis, offline analysis, interactive query, real-time retrieval, multi-modal analysis, data warehousing, and data ingestion and governance of full data of government and enterprise customers.

9.2 Components

9.2.1 Hudi

9.2.1.1 Introduction to Hudi

Hudi is an open-source project launched by Apache in 2019 and became a top Apache project in 2020. Huawei participated in Hudi community development in 2020 and used Hudi in FusionInsight.

Hudi is the file organization layer of the data lake. It manages Parquet files, provides data lake capabilities and IUD APIs, and supports compute engines.

9.2.1.2 Hudi Features

Hudi features:

- Supports real-time and batch data import into the lake with ACID capabilities.
- Provides a variety of views (read-optimized view/incremental view/real-time view) for quick data analysis.
- Uses the multi-version concurrency control (MVCC) design and supports data version backtracking.
- Automatically manages file sizes and layouts to optimize query performance and provide quasi-real-time data for queries.
- Supports concurrent read and write. Data can be read when being written based on snapshot isolation.
- Supports bootstrapping to convert existing tables into Hudi datasets.

Key technologies and advantages of Hudi:

- Pluggable index mechanism: Hudi provides multiple index mechanisms to quickly update and delete massive data.
- Ecosystem support: Hudi supports multiple data engines, including Hive, Spark, and Flink.

9.2.1.3 Hudi Architecture: Batch and Real-Time Data Import, Compatibility with Diverse Components, and Open-Source Storage Formats

Hudi datasets (data organization mode of Hudi tables)

- Data files: base files and delta log files (corresponding to the update and delete operations on base files)
- Index: Bloom filter index generated based on the primary key of the Hudi table. The default level is file groups.
- Timeline metadata: manages data commit logs to implement snapshot isolation, views, and rollback.

Views:

- Read-optimized view (for efficient read): The base file generated after compaction is read. The data that is not compacted is not the latest.

- Incremental view (for frequent updates): The latest data is read. The base file and delta files are merged during read.
- Real-time view (for stream-batch convergence): Data that is incrementally written into Hudi is continuously read in a way similar to CDC.

9.2.2 HetuEngine

9.2.2.1 Introduction to HetuEngine

HetuEngine is a self-developed high-performance engine for interactive SQL analysis and data virtualization. It seamlessly integrates with the big data ecosystem to implement interactive query of massive amounts of data within seconds, and supports cross-source and cross-domain unified data access to enable one-stop SQL convergence analysis in the data lake, between lakes, and between lakehouses.

9.2.2.2 HetuEngine Application Scenarios

HetuEngine supports interactive quick query across sources (multiple data sources, such as Hive, HBase, GaussDB(DWS), Elasticsearch, and ClickHouse) and across domains (multiple regions or data centers), especially for Hive and Hudi data of Hadoop clusters (MRS).

- Cross-domain:
  - Cross-domain service entry
  - Distributed networking
  - High-performance cross-domain data transmission at GB/s level
  - Zero metadata synchronization
  - Restricted data access
  - Cross-domain computing pushdown
  - Fast rollout with simplified configuration
- Cross-source:
  - Cross-domain service entry
  - Distributed networking
  - High-performance cross-domain data transmission at GB/s level
  - Zero metadata synchronization
  - Restricted data access
  - Cross-domain computing pushdown
  - Fast rollout with simplified configuration
- Cloud-native:
  - Cloud service entry
  - Centralized O&M of resources and permissions
  - Visualized and instant data source configuration
  - Auto scaling
  - Multi-instance and multi-tenant deployment

- Rolling restart without interrupting services
- Backup & DR

9.2.3 Ranger

9.2.3.1 Ranger

Apache Ranger offers a centralized security management framework and supports unified authorization and auditing. It manages fine-grained access control over Hadoop and related components, such as HDFS, Hive, HBase, Kafka, and Storm. Users can use the front-end web UI provided by Ranger to configure policies to control users' access to these components.

9.2.3.2 Ranger Architecture

Ranger consists of the following components:

- RangerAdmin: provides a web UI and RESTful APIs to manage policies, users, and auditing.
- UserSync: periodically synchronizes user and user group information from an external system and writes the information to RangerAdmin.
- TagSync: periodically synchronizes tag information from the external Atlas service and writes the tag information to RangerAdmin.

9.2.3.3 Ranger Principles

- Ranger Plug-ins
  - Ranger provides Policy-Based Access Control (PBAC) plug-ins to replace the original authentication plug-ins of the components. Ranger plug-ins are developed based on the authentication interface of the components. Users set permission policies for specified services on the Ranger web UI. Ranger plug-ins periodically update policies from the RangerAdmin and caches them in the local file of the component. When a client request needs to be authenticated, the Ranger plug-in matches the user carried in the request with the policy and then returns an accept or reject message.
- UserSync user synchronization
  - UserSync periodically synchronizes data from LDAP (security mode) or Unix (non-security mode) to RangerAdmin. By default, the incremental synchronization mode is used. In each synchronization period, UserSync updates only new or modified users and user groups. When a user or user group is deleted, UserSync does not synchronize the change to RangerAdmin. That is, the user or user group is not deleted from the RangerAdmin. To improve performance, UserSync does not synchronize user groups to which no user belongs to RangerAdmin.
- Unified Auditing
  - Ranger plug-ins can record audit logs. Currently, audit logs can be stored in local file systems.
- Reliability

- Ranger supports two RangerAdmins working in active/active mode. Two RangerAdmins provide services at the same time. If either RangerAdmin is faulty, Ranger continues to work.

- High Performance
  - Ranger provides the load balancing capability. When a user accesses Ranger web UI using a browser, the Load-Balance automatically selects the RangerAdmin with the lightest load to provide services.

9.2.4 LDAP

In 1988, the International Telegraph and Telephone Advisory Committee developed the X.500 standard and updated it in 1993 and 1997. The X.500 standard features perfect function design and flexible scalability. It defines the comprehensive directory service, including the information model, namespace, function model, access control, directory replication, and directory protocol. It quickly becomes the standard that all directory servers comply with. X.500 defines the Directory Access Protocol (DAP) protocol for communication between a client and a server of a directory service. However, due to the complex DAP structure and strict compliance with the OSI seven-layer protocol model during running, it is difficult to deploy DAP in many small environments, and it usually runs in UNIX environments. Therefore, the University of Michigan launched a TCP/IP-based Lightweight Directory Access Protocol (LDAP) based on the X.500 standard.

The LDAP client and server communicate with each other independently. Users can access various LDAP servers through different LDAP clients. Currently, LDAP has two versions: LDAP v2 and LDAP v3. Most directory servers comply with LDAP v3 to provide various query services for users. LDAP is just a protocol and does not involve data storage. Therefore, a backend database component is required to implement storage. These backends can be Berkeley DB, shell, passwd, or others.

LDAP has the following characteristics:

- Flexibility: In LDAP, the Access Protocol (AP) is not only an X.500-based access protocol, but also a flexible directory system that can be implemented independently.
- Cross-platform: LDAP is a cross-platform and standard-based protocol and can serve various operating systems such as Windows, Linux, and Unix.
- Low cost and easy configuration management: LDAP runs on TCP/IP or other connection-oriented transmission services. Most LDAP servers are easy to install and configure, respond quickly, and seldom require maintenance during long-term use.
- Access Control List (ACL): LDAP can use ACLs to control access to directories. For example, an administrator can specify the permissions of members in a given group or location, or grant specific users to modify selected fields in their own records.
- LDAP is an Internet Engineering Task Force (IETF) standard track protocol and is specified in *RFC 4510 on Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*.
- Distributed: LDAP manages physically distributed data in a unified manner and ensures logical integrity and consistency of the data. LDAP implements distributed operations through client APIs, thereby balancing load.

As a directory service system, the LDAP server provides centralized user account management. It has the following advantages:

- When facing a large, fast-growing number of users, the LDAP server makes it easier for you to manage user accounts, specifically, to create accounts, reclaim accounts, manage permissions, and audit security.
- The LDAP server makes it secure to access different types of systems and databases across multiple layers. All account-related management policies are configured on the server, implementing centralized account maintenance and management.
- The LDAP server fully inherits and utilizes the identity authentication functions of existing account management systems on the platform. It separates account management from access control, thus improving access authentication security of the big data platform.

On the Huawei big data platform, an LDAP server functions as a directory service system to implement centralized account management. An LDAP server is a directory service system that consists of a directory database and a set of access protocols. It has the following characteristics:

- The LDAP server is based on the open-source OpenLDAP technology. (OpenLDAP project is built by a team of volunteers.)
- The LDAP server uses Berkeley DB as the default backend database.
- The LDAP server is an open-source implementation of the LDAP protocol.

The LDAP server directory service system consists of basic models such as organizational models and function models.

9.2.5 Kerberos

Kerberos is an authentication concept named after the ferocious three-headed guard dog of Hades from Greek mythology. The Kerberos protocol adopts a client – server model and cryptographic algorithms such as Data Encryption Standard (DES) and Advanced Encryption Standard (AES). Furthermore, it provides mutual authentication, so that the client and server can verify each other's identity. As a trusted third-party authentication service, Kerberos performs identity authentication based on shared keys.

The Kerberos protocol was developed by Massachusetts Institute of Technology and was originally provided as a network server to protect the Athena project. Steve Miller and Clifford Neuman released Kerberos version 4 at the end of 1980, mainly for the Athena project. In 1993, John Kohl and Clifford Neuman released Kerberos version 5, which resolves limitations and security issues on the basis of version 4. In 2005, the Kerberos working group of the Internet Engineering Task Force updated the protocol specifications. Currently, Kerberos version 5 is the mainstream network identity authentication protocol. Windows 2000 and later operating systems use Kerberos as the default authentication method. Apple Mac OS X uses the Kerberos client and server versions. Red Hat Enterprise Linux 4 and later operating systems use the Kerberos client and server versions.

Huawei big data platform builds the KrbServer identity authentication system based on the Kerberos protocol and provides security authentication functions for all open-source components to prevent eavesdropping, replay attacks, and data integrity. It is a system that manages keys by using a symmetric key mechanism.

Kerberos has the following advantages:

- Prevents brute-force attacks. The session key used for authentication is a short term key, which is valid only in one session.
- Prevents replay attacks. Each request is marked with a timestamp.
- Supports mutual authentication. Kerberos supports mutual authentication, which is better than NTLM. The Kerberos server returns the timestamp sent by the client for identity verification.
- Provides high performance. The KRB_AS_REP and KRB_TGS_REP messages show that KDC sends the identity information of the client, including the session key and encrypted client identity information (TGT and session ticket), to the client, which stores the information. In this way, the storage pressure of the server and KDC is reduced. The client instead of the KDC sends the authenticator and session ticket in a unified manner. This reduces the KDC pressure and prevents unsuccessful authentication caused by the failure of the client and KDC to reach the server simultaneously.

Kerberos has the following defects:

- In the AS exchange process, AS uses the master key of the client to encrypt the response to the request sent to the client. Therefore, the master key of the client is still used for encryption and transmission. The TGT granted by the AS to the client has a lifetime. When the lifetime ends, the client must resend the KRB_AS_REQ message to the AS. As a result, the master key is transmitted multiple times on the network.
- The KDC needs to store a large number of keys, causing high costs of maintaining its account database.
- Kerberos uses timestamps to prevent replay attacks. However, in a distributed system, strict time synchronization is difficult, and incorrect time synchronization causes an authentication failure. This allows attackers to launch attacks by interfering with the time system.

9.2.6 Architecture of Huawei Big Data Security Authentication Scenarios

To manage the access control permissions on data and resources in a cluster, it is recommended that the cluster on the Huawei big data platform be installed in security mode. In security mode, a client application must be authenticated and a secure session must be established before the application accesses any resource in the cluster.

Huawei big data platform provides three security authentication scenarios.

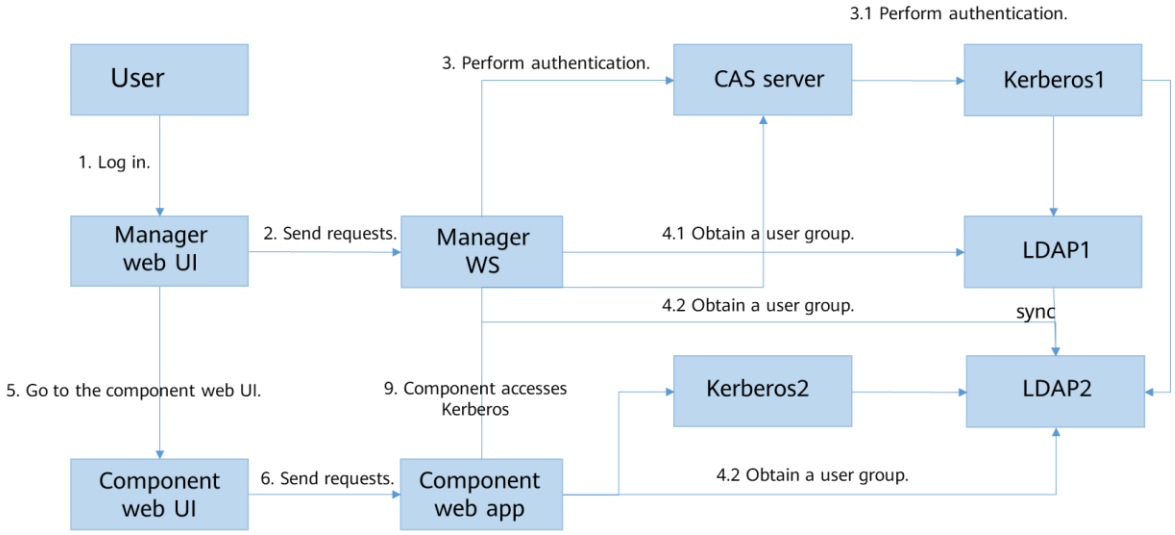


Figure 9-1 Architecture of Huawei security authentication scenarios

- **Process for logging in to the Manager web UI**: Steps 1, 2, 3, and 4 in the architecture. After a user logs in to the Manager web UI, the web UI sends a request to the Manager web browser, and then the CAS server sends an identity authentication request to KrbServer. KrbServer verifies the user identity from the LDAP server. After the verification is successful, KrbServer can obtain the required user group from the LDAP server.
- **Process for logging in to the component web UI**: Steps 5, 6, 7, and 8 in the architecture. After a user logs in to the component web UI, the web UI sends a request to the component web app, and then the CAS server sends an identity authentication request to KrbServer. KrbServer verifies the user identity from the LDAP server. After the verification is successful, KrbServer can obtain the required user group from the LDAP server.
- **Security authentication for access between components**: Step 9 in the architecture. The Kerberos service is used to implement security authentication between different open-source components on Huawei big data platform.

Kerberos1 operations on LdapServer data:

Active and standby instances of LdapServer1 and LdapServer2 are accessed in load sharing mode. Data can be written only in the active LdapServer1 instance. Data can be read on LdapServer1 or LdapServer2.

Kerberos2 operations on LdapServer data: Only the active and standby instances of LdapServer2 can be accessed. Data can be written only in the active LdapServer2 instance.

9.3 MRS Cloud-Native Data Lake Baseline Solution

9.3.1 Panorama of the FusionInsight MRS Cloud-Native Data Lake Baseline Solution in Huawei Cloud Stack

Huawei Cloud MRS cloud-native data lake offers a lakehouse architecture that is capable of continuous evolution and can be used to build three different types of data lakes: offline, real-time, and logical. It applies to real-time analysis, offline analysis, interactive query, real-time retrieval, multi-modal analysis, data warehousing, data ingestion, and data governance. It enables enterprises to use their data in an efficient and simple way, achieve "one lake for one enterprise" and "one lake for one city", gain accurate insight into business, and monetize data more quickly.

9.3.2 Offline Data Lake

Offline data lake: provides multiple computing engines such as interactive, BI, and AI engines, and uses OBS to separate storage from compute, making the architecture of the cloud-native data lake more flexible. Supports ultra-large scale of 20,000+ nodes in a single cluster and 100,000+ nodes through cluster federation. Supports rolling upgrade to ensure uninterrupted upgrade of key services.

Offline data lakes are also called lakehouses in many vendors. A data warehouse is a concept in the database era. It refers to an offline data lake that supports only SQL. The offline data lake has stronger capabilities than the data warehouse. In addition to SQL, the offline data lake supports data mining, AI analytics, offline analytics, and interactive query.

The database can be loaded in real time by using the CDC synchronization tool, thereby achieving the real-time data lake. Streaming data, such as IoT data, is processed by the stream processing engine (such as Flink), accumulated as batches, and then imported into the lake. It is still an offline data lake. The time from data generation to data import into the lake is the standard for distinguishing offline and real-time data lakes.

The offline data lake contains the interactive query engine. Although data cannot be imported to the lake in real time, data imported to the lake in batches can still be queried within seconds.

Offline data lake = Original batch processing solution + Original real-time stream processing solution + Original interactive query solution

Specialized data mart: stores data in a specific format, used for specific types of query or analytics or for specific users. Different data marts may have duplicate data.

Real-time and quasi-real-time: Data can be stored in the data lake within 15 minutes after being generated from data sources.

Core requirements of the offline data lake:

- Large-scale cluster: The cluster must have more than 100 nodes to handle more than 1 PB of data.
- High-concurrency interactive query: Data in the data lake can be queried within 2 seconds under hundreds of concurrent queries.

- Intra-lake update operation: In addition to common query and append operations, update operations are also supported during offline data processing, that is, lakehouse.
- One replica of data storage: Only one replica of data is stored to support multiple types of analysis, offline processing, and interactive query.
- Data permission control and resource isolation (multi-tenant): Multiple offline processing jobs run at the same time. Different data permissions and resource scheduling policies are required to prevent unauthorized access or resource preemption.
- Compatibility with open-source APIs: Customers usually have inventory offline processing applications that need to be migrated to the offline data lake.
- Multiple data sources and data loading methods: Data from different sources is of different types and stored in different formats.
- Rolling upgrade: Offline processing is the basis of the customer's big data system. The customer demands that the system is upgraded without service interruptions.
- Job scheduling management: Multiple offline jobs have different priorities and different running time. Therefore, multiple scheduling policies are required to monitor abnormal and failed jobs.
- Heterogeneous devices: During capacity expansion, customers can configure the devices to be upgraded and use the old and new devices separately.
- Interconnection with third-party software (visualization, analysis and mining, report, and metadata): Multiple third-party tools can be interconnected to facilitate further data analysis and management.

9.3.3 Real-Time Data Lake

Real-time data lake: builds real-time update and processing capabilities through Hudi's real-time incremental import of ACID data into the lake and ClickHouse's millisecond-level OLAP analysis, improving data timeliness from T+1 to T+0.

The real-time data lake is an upgrade of the offline data lake. It adds the real-time data ingestion capability to the offline data lake and supports the batch-stream integration engine. The two are the same data lake. Some vendors call it "real-time data warehouse".

The database can be loaded in real time by using the CDC synchronization tool, thereby achieving the real-time data lake. Streaming data is processed by the stream processing engine (such as Flink), accumulated as batches, and then imported into the lake. It is still an offline data lake. The time from data generation to data import into the lake is the standard for distinguishing offline and real-time data lakes.

Customers may analyze data in various scenarios. Some scenarios have special requirements, such as the graph and time series databases, some require ultra-high performance, such as the real-time OLAP and in-memory library, and some need to consider existing applications, such as the search library. Therefore, specialized marts are still an important supplement to the data lakes.

Real-time data lake = Original batch processing solution + Original real-time stream processing solution + Original interactive query solution + New Flink SQL batch-stream integration engine

The real-time data lake and offline data lake are the same data lake. Only one copy of data is stored, which is the basic requirement of the real-time data lake. Many vendors store data both in real time and offline. Although the data is stored in the same HDFS, there are still two lakes for users. For example, in the vendor's solution of the MaxCompute offline data lake and E-MapReduce real-time data lake, there are two lakes and two copies of data are stored. For users, the real-time data lake is mandatory, and the specialized mart is optional. The real-time data lake can meet the requirements, and there's no need to build a specialized mart.

Core requirements of the real-time data lake:

- Large-scale cluster: The cluster must have more than 100 nodes to handle more than 1 PB of data.
- High-concurrency interactive query: Data in the data lake can be queried within 2 seconds under hundreds of concurrent queries.
- Intra-lake update operation: In addition to common query and append operations, update operations are also supported during offline and real-time data processing, that is, lakehouse.
- One replica of data storage: Only one replica of data is stored to support multiple types of analysis. Offline data lakes and real-time data lakes cannot store multiple replicas of data.
- Data permission control and resource isolation (multi-tenant): Multiple offline and real-time processing jobs run at the same time. Different data permissions and resource scheduling policies are required to prevent unauthorized access or resource preemption.
- Compatibility with open-source APIs: Customers usually have inventory processing applications that need to be migrated to the real-time data lake.
- Rolling upgrade: Offline and real-time processing is the basis of the customer's big data system. The customer demands that the system is upgraded without service interruptions.
- Job scheduling management: Multiple offline and real-time jobs have different priorities and different running time. Therefore, multiple scheduling policies are required to monitor abnormal and failed jobs.
- Heterogeneous devices: During capacity expansion, customers can configure the devices to be upgraded and use the old and new devices separately.
- Interconnection with third-party software (visualization, analysis and mining, report, and metadata): Multiple third-party tools can be interconnected to facilitate further data analysis and management.
- Real-time data import into the lake: It takes less than 15 minutes from data generation to import into the lake.
- Multiple offline and real-time data sources, including traditional file, database synchronization, and message queue data.

9.3.4 Logical Data Lake

Logical data lake: uses HetuEngine to provide cross-lake, cross-warehouse, and cross-cloud collaborative analysis and implement lakehouses, cutting the data migration time by 80% and speeding up collaborative analysis by 50 times.

A logical data lake means that data is scattered on multiple data platforms. The timeliness of joint analysis is far worse than that of an offline or real-time data lake. Therefore, a logical data lake is an effective supplement to an offline or real-time data lake when data cannot be centralized due to certain practical reasons (such as laws, security, and costs).

Whether to migrate data depends on the analysis mode. For the analysis in which data is dispersedly managed, such as data retrieval, data does not need to be migrated. For the analysis in which data must be combined, such as multi-table associated analysis, data needs to be migrated. The logical data lake can achieve only "computing with less data migration".

A logical data lake is a virtual data lake comprising multiple offline or real-time data lakes. Its security depends on the security policies of the offline or real-time data lakes. In addition, when an offline or real-time data lake is accessed through a logical data lake, data flows are strictly controlled to prevent unauthorized access. Therefore, using a logical data lake is more secure than connecting data lakes by using applications.

Offline or real-time data lake: an offline or real-time data lake, or simply called a data lake, is a physical data platform that aggregates data.

Core requirements of the logical data lake:

- Different data lakes, databases, and data marts: Logical data lakes can be built based on multiple data lakes, databases, and data marts.
- High cross-lake analysis performance: The performance of cross-lake analysis, especially complex cross-lake analysis, can match different network conditions between lakes and optimize the query performance.
- High data security of cross-lake analysis: Cross-lake analysis cannot change the original security policies of each lake. In addition, cross-lake analysis must comply with the security policies of each lake and cannot bring new security problems.
- Consistent usage and management: The usage and management of the logical data lake should be consistent with those of the offline or real-time data lake, eliminating impact on user experience.
- Global data view: The logical data lake solution provides a unified global data view for customers, allowing administrators and users to manage and use data in the logical data lake under the constraints of the security policy framework.

9.3.5 X Bank: Rolling Upgrade, Storage-Compute Decoupling, and HetuEngine-based Real-Time BI

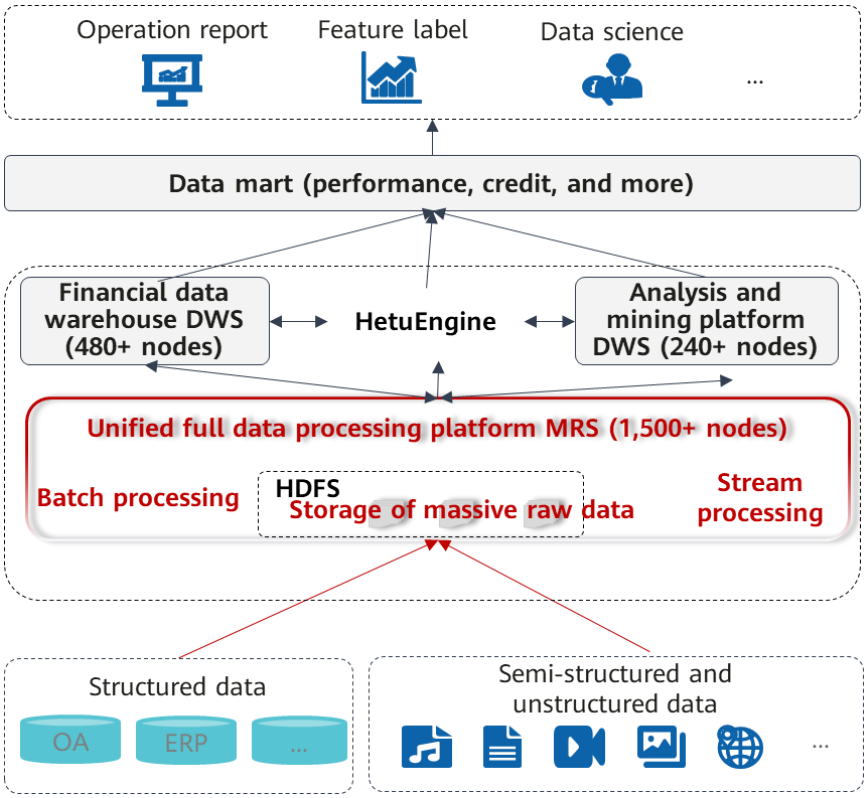


Figure 9-2 ICBC big data system

Recently, X bank has been using FusionInsight to build its big data system. Today, this big data platform contains more than 1,500 nodes. This platform batch processes more than 100,000 jobs and stores more than 30 petabytes of data each day. X's legacy systems cannot keep up with its rapidly growing data volume. A traditional upgrade solution usually requires multiple power-offs and restarts, which interrupt mission-critical services such as anti-fraud and precision marketing. In addition, because compute and storage resources are usually tightly coupled, they need to be scaled up or down together, and since these two types of resources are typically not equally utilized, doing so may lead to even lower utilization of computing resources. MRS supports rolling upgrade, where cluster nodes can be upgraded in different batches and faulty nodes can be isolated automatically. This ensures zero service downtime during the upgrade. The application systems are not even aware of the upgrade.

MRS' storage-compute decoupled architecture improves compute resource utilization by 30%. HetuEngine enables collaborative analysis across data lakes and warehouses, reducing unnecessary ETL operations caused by the separation between the lake and warehouse and also improving analytical efficiency. According to past statistics, ETL operations are reduced by 80%, and analytical efficiency is improved by more than 10 times.

9.3.6 XX Healthcare Security Administration: Builds a Unified Offline Data Lake for Decision-Making Support

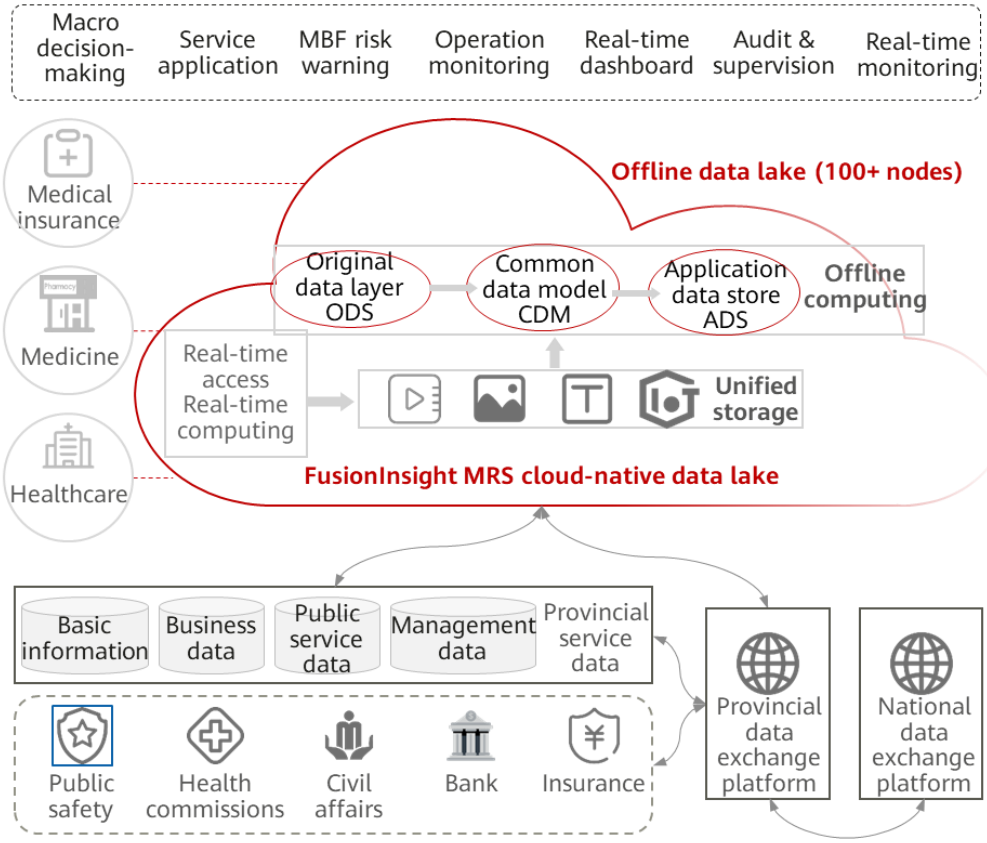


Figure 9-3 Offline healthcare security data lake

Customer pain points:

- The healthcare security, medicine, and medical treatment systems are siloed. A unified data platform covering all subjects, objects, services, processes, and data is needed.
- Scattered management and fixed standards cause low efficiency in business handling and operation.
- It is difficult to detect and prevent violations in medical insurance reimbursement and insurance fraud.

Solution:

- Use MRS to build an offline data lake to store data from different sources, such as medical insurance, medication, and medical treatment. Build the original data store (ODS), common data model (CDM), and application data store (ADS) in the offline and real-time computing areas to implement comprehensive data management and governance and build a unified medical insurance data platform for the entire province.
- Use unified data standards to implement centralized data governance and correlated analysis of data from different sources.

- Build an offline data lake to centrally store video, image, text, and IoT data, providing a real-time computing area and real-time data processing capabilities.

Customer benefits:

- A one-stop platform that holds all-domain data is provided for people to handle healthcare business. The intensive construction reduces data silos and TCO by 30%.
- The medical insurance reimbursement efficiency is increased by 3 times, and the manual review workload and error rate are decreased by 80%. People only need to visit the office once to handle a single business.
- The real-time data computing capability effectively controls vulnerabilities that may breed medical insurance reimbursement violations and insurance fraud, recovering XX00 million economic loss every year and ensuring sound development of the medical benefits fund (MBF).

9.4 Quiz

What are the advantages of MRS compared with self-built Hadoop?

10 Huawei DataArts Studio

10.1 Data Governance

Challenges to Enterprise Digital Transformation

Enterprises often face challenges in the following aspects when managing data:

- Governance
  - Inconsistent data system standards impact data exchange and sharing between different departments.
  - There are no great search tools to help service personnel locate the data they need when they need it.
  - If metadata fails to define data in business terms that are familiar to data consumers, the data is difficult to understand.
  - When there are no good methods to evaluate and control data quality, it makes the data hard to trust.
- Operations
  - Data analysts and decision makers require efficient data operations. There is no efficient data operations platform to address the growing and diversified demands for analytics and reporting.
  - Repeated development of the same data wastes time, slows down development, and results in too many data copies. Inconsistent data standards waste resources and drive up costs.
- Innovation
  - Data silos prevent data from being shared and circulated across departments in enterprises. As a result, cross-domain data analysis and data innovation fail to be stimulated.
  - Currently, most enterprises still utilize their data for analytics and reporting. There is a long way to go before enterprises have widespread, data-driven service innovation.

10.2 DataArts Studio

10.2.1 What is DataArts Studio?

DataArts Studio is a one-stop data operations platform that drives digital transformation. It allows users to perform many operations, such as integrating and developing data, designing data architecture, controlling data quality, managing data assets, creating data services, and ensuring data security. Incorporating big data storage, computing and analytical engines, it can also construct industry knowledge bases and help enterprises build an intelligent end-to-end data system. This system can eliminate data silos, unify data standards, accelerate data monetization, and accelerate enterprises' digital transformation.

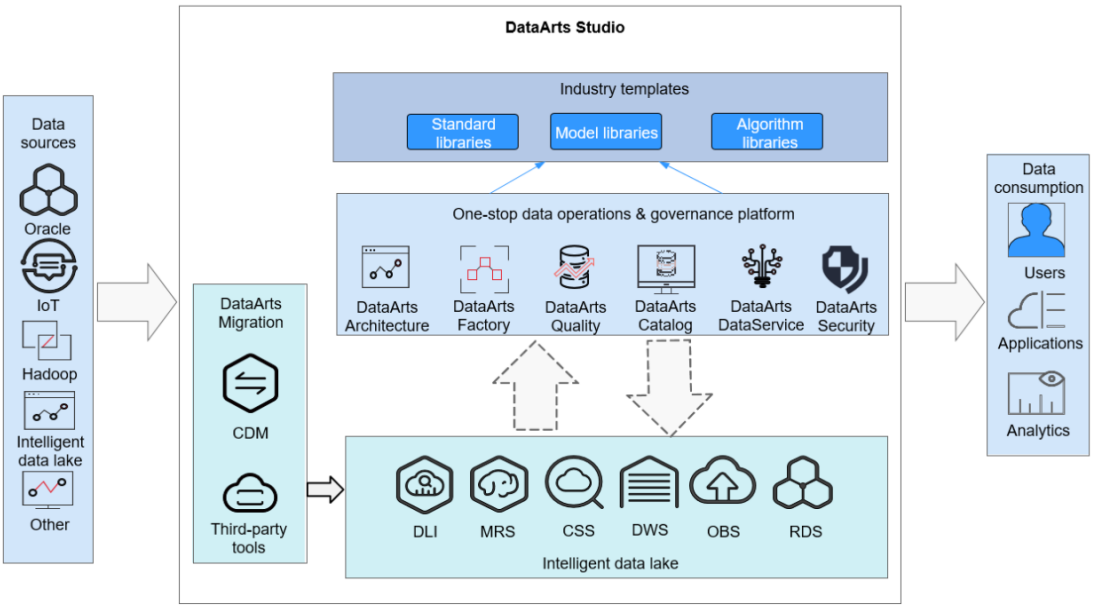


Figure 10-1 DataArts Studio architecture

As shown in the figure, DataArts Studio is based on the data lake base and provides capabilities such as data integration, development, governance, and openness. DataArts Studio can connect to Huawei Cloud data lakes and cloud database services, such as Data Lake Insight (DLI), MRS Hive, and GaussDB(DWS). These data lakes and cloud database services are used as the data lake base. DataArts Studio can also connect to traditional enterprise data warehouses, such as Oracle and Greenplum.

DataArts Studio consists of the following components:

- Management Center

  Management Center supports data connection management and connects to the data lake base for activities such as data development and data governance.

- DataArts Migration

  DataArts Migration supports data migration between 20+ data sources and integration of data sources into the data lake. It provides wizard-based configuration and management and supports single table, entire database, incremental, and periodic data integration.

- DataArts Architecture

  DataArts Architecture helps users plan the data architecture, customize models, unify data standards, visualize data modeling, and label data. DataArts Architecture defines how data will be processed and utilized to solve business problems and enables users to make informed decisions.

- DataArts Factory

  DataArts Factory helps users build a big data processing center, create data models, integrate data, develop scripts, and orchestrate workflows.

- DataArts Quality

  DataArts Quality monitors the data quality in real time with data lifecycle management and generates real-time notifications on abnormal events.

- DataArts Catalog

  DataArts Catalog provides enterprise-class metadata management to clarify information assets. A data map displays data lineages and data assets for intelligent data search, operations, and monitoring.

- DataArts DataService

  DataArts DataService is a platform where users can develop, test, and deploy their data services. It ensures agile response to data service needs, easier data retrieval, better experience for data consumers, higher efficiency, and better monetization of data assets.

- DataArts Security

  DataArts Security provides all-round protection for users' data. Users can use it to detect sensitive data, grade and classify data, safeguard data privacy, control data access permissions, encrypt data during transmission and storage, and identify data risks. DataArts Security is an efficient tool for users to establish security warning mechanisms. It can greatly improve users' overall security protection capability, securing their data and making their data more accessible.

10.2.2 DataArts Migration: Efficient Ingestion of Multiple Heterogeneous Data Sources

DataArts Migration can help users seamlessly migrate batch data between more than 30 homogeneous or heterogeneous data sources. It supports multiple on-premises and cloud-based data sources, including file systems, relational databases, data warehouses, NoSQL databases, big data cloud service, and object storage.

DataArts Migration uses a distributed compute framework and concurrent processing techniques to help users migrate enterprise data in batches without any downtime and rapidly build desired data structures.

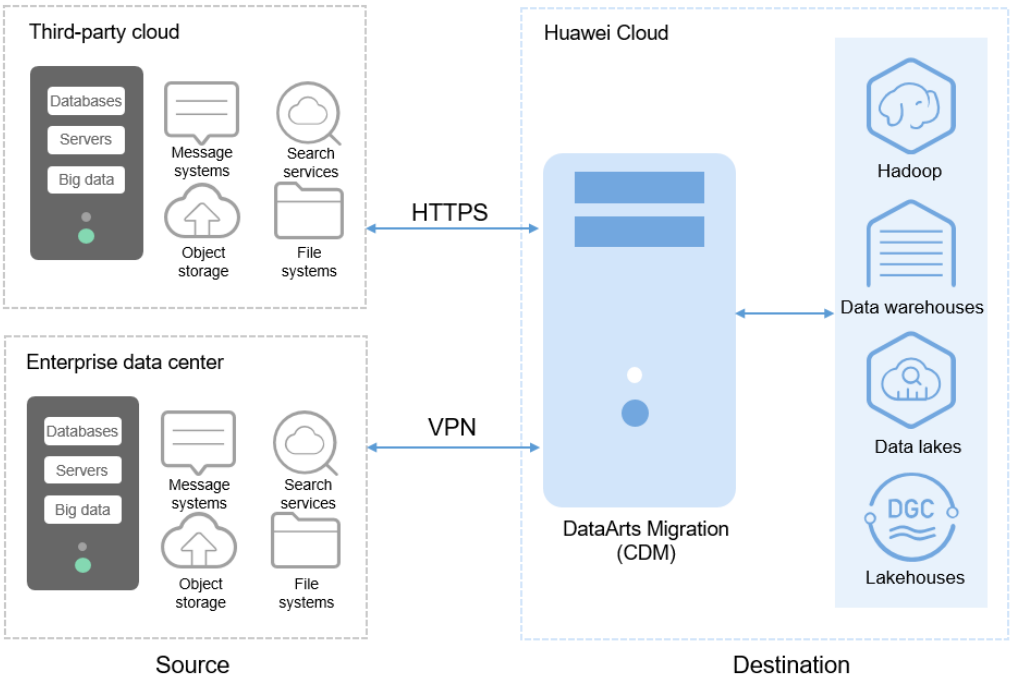


Figure 10-2 DataArts Migration

Users can manage data on the wizard-based task management page and easily create data migration tasks that meet their requirements. DataArts Migration provides the following functions:

- Table/File/Entire DB migration

  Tables or files can be migrated in batches. An entire database can be migrated between homogeneous and heterogeneous databases. A job can migrate hundreds of tables.

- Incremental data migration

  DataArts Migration supports incremental migration of files, relational databases, and HBase. It also supports incremental migration through WHERE clauses and macro variables of date and time.

- Migration in transaction mode

  If a migration job fails, DataArts Migration rolls back the data to the state before the job started and automatically deletes data from the destination table.

- Field conversion

  DataArts Migration supports data conversion, including anonymization, string operations, and date operations.

- File encryption

  Users can encrypt files that are migrated to a cloud-based file system in batches.

- MD5 verification

  MD5 is used to check file consistency from end to end, and the check result is generated.

- Dirty data archiving

DataArts Migration can automatically archive the data that fails to be processed during migration, has been filtered out, or is not compliant with conversion or cleaning rules to dirty data logs for users to analyze. A threshold for dirty data ratio can be set to determine whether a task is successful.

10.2.3 DataArts Architecture: Visualized, Automated, and Intelligent Data Modeling

DataArts Architecture incorporates data governance methods. You can use it to visualize data governance operations, connect data from different layers, formulate data standards, and generate data assets. You can standardize your data through ER modeling and dimensional modeling. DataArts Architecture is a good option for unified construction of metric platforms. With DataArts Architecture, you can build standard metric systems to eliminate data ambiguity and facilitate communications between different departments. In addition to unifying computing logic, you can use it to query data and explore data value by subject.

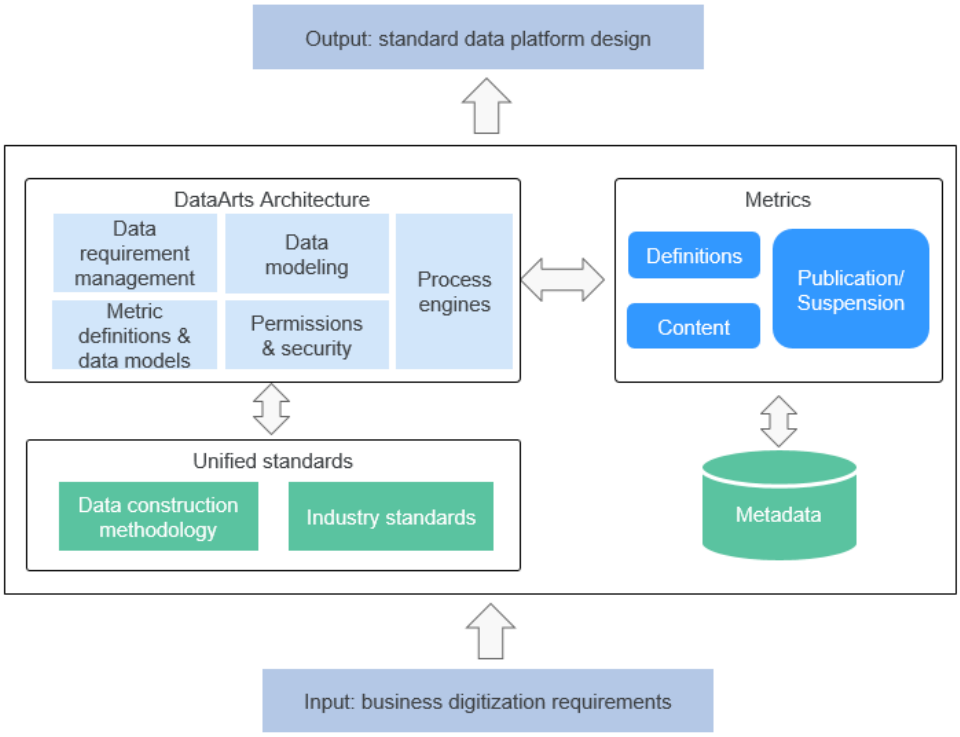


Figure 10-3 DataArts Architecture

DataArts Architecture provides the following major functions:

- Subject design

  DataArts Architecture can be used to build unified data classification systems for directory-based data management. Data classification, search, evaluation, and usage are easier than ever before. DataArts Architecture provides hierarchical architectures that help users define and classify data assets, allowing data consumers to better understand and trust users' data assets.

- Data standards

DataArts Architecture can help users create process-based and systematic data standards that fit their needs. Peered with the national and industry standards, these standards enable users to standardize their enterprise data and improve data quality, ensuring that their data is trusted and usable.

- Data modeling

  Data modeling involves building unified data model systems. DataArts Architecture can be used to build a tiered, enterprise-class data system based on data specifications and models. The system incorporates data from the public layer and subject libraries, significantly reducing data redundancy, silos, inconsistency, and ambiguity. This allows freer flow of data, better data sharing, and faster innovation.

  DataArts Architecture provides the following data modeling methods:

- ER modeling

  Entity Relationship (ER) modeling involves describing the business activities of an enterprise, and ER models are compliant with the third normal form (3NF). ER models can be used for data integration, which merges and classifies data from different systems by similarity or subject. However, ER models cannot be used for decision-making.

- Dimensional modeling

  Dimensional modeling involves constructing bus matrices to extract business facts and dimensions for model creation. Users need to sort out business requirements for constructing metric systems and creating summary models.

10.2.4 DataArts Factory: One-stop Collaborative Development

DataArts Factory is a one-stop agile big data development platform. It provides a visualized graphical development interface, rich data development types (script development and job development), fully-hosted job scheduling and O&M monitoring capabilities, built-in industry data processing pipeline, one-click development, full-process visualization, and online collaborative development by multiple people, as well as supports management of multiple big data cloud services, greatly lowering the threshold for using big data and helping you quickly build big data processing centers.

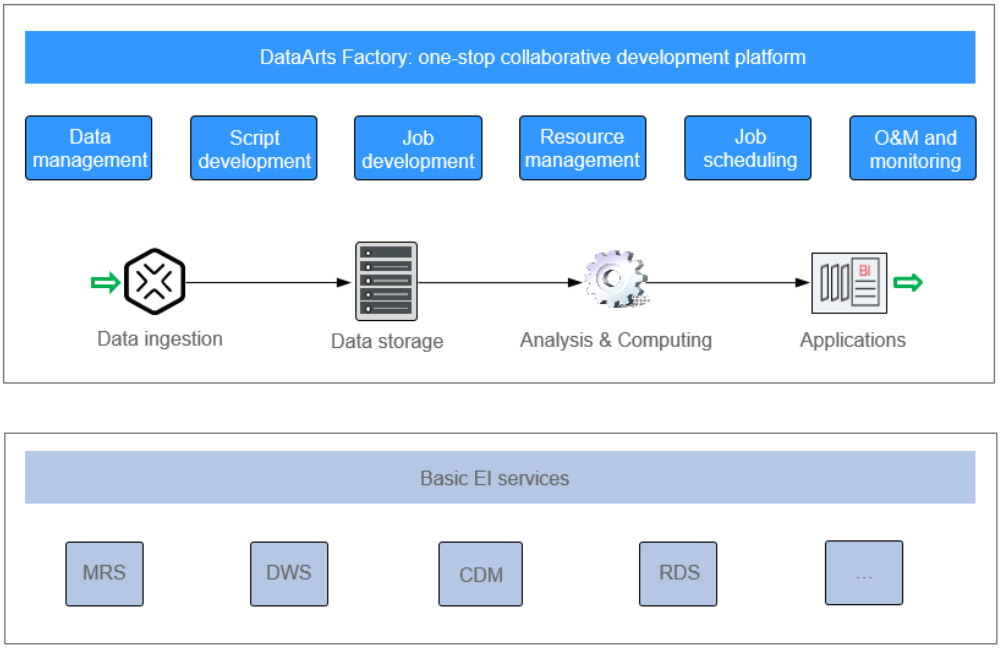


Figure 10-4 DataArts Factory architecture

DataArts Factory allows users to manage data, develop scripts, and schedule and monitor jobs. Data analysis and processing are easier than ever before.

- Data management

  Users can manage multiple types of data warehouses, such as GaussDB(DWS), DLI, and MRS Hive.

  Users can use the graphical interface and data definition language (DDL) to manage database tables.

- Script development

  DataArts Factory provides an online script editor that allows multiple users to collaboratively develop and debug SQL, Python, and Shell scripts.

  Variables are supported.

- Job development

  DataArts Factory provides a graphical designer that allows users to rapidly develop workflows through drag-and-drop and build data processing pipelines.

  DataArts Factory is preset with multiple task types such as data integration, SQL, Spark, Shell, and machine learning. Data is analyzed and processed by dependency between tasks.

  Job import and export are supported.

- Resource management

  Users can centrally manage file, jar, and archive resources used during script and job development.

- Job scheduling

  Users can schedule jobs to run once or recursively and use events to trigger scheduling jobs.

Job scheduling supports a variety of hybrid orchestration tasks of cloud services. The high-performance scheduling engine has been verified by hundreds of applications.

- O&M and monitoring

  Users can run, suspend, restore, and terminate a job.

  Users can view the details of each job and each node in the job.

  Users can use various methods to receive notifications when a job or task error occurs.

10.2.5 DataArts Quality: Verifiable and Controllable Data Quality

DataArts Quality can monitor metrics and data quality, and screen out unqualified data in a timely manner.

- Metric monitoring

  DataArts Quality can be used to monitor the quality of data in databases. Users can create metrics, rules, or scenarios that meet their requirements and schedule them in real time or recursively.

- Data quality monitoring

  Users can create data quality rules to check whether the data in their databases is accurate in real time.

  Qualified data must meet the following requirements: integrity, validity, timeliness, consistency, accuracy, and uniqueness. Users can standardize data and periodically monitor data across columns, rows, and tables based on quality rules.

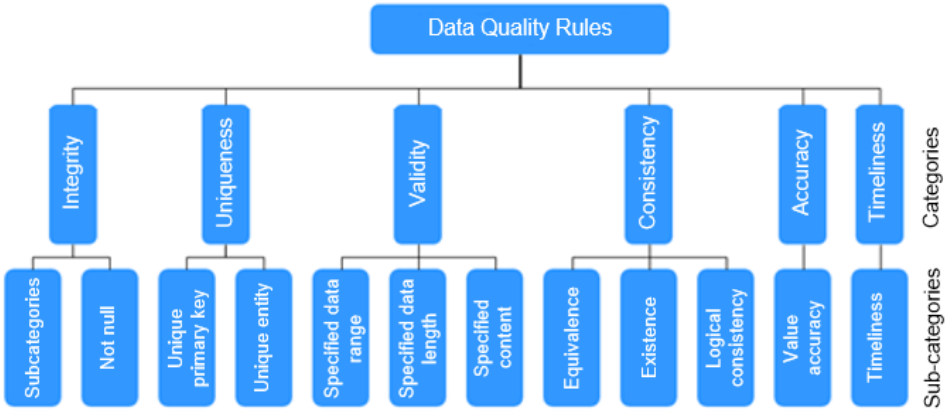


Figure 10-5 Data quality rule system

DataArts Studio provides enterprise-class metadata management to clarify information assets. It also supports data drilling and source tracing. It uses a data map to display a data lineage and panorama of data assets for intelligent data search, operations, and monitoring.

- Metadata management

  The metadata management module is the cornerstone of data lake governance. It supports the creation of collection tasks with custom policies and can collect technical metadata from data sources. It also allows users to customize a business

metamodel to batch import business metadata, associate business metadata with the technical metadata, and manage and apply linkages throughout the entire link.

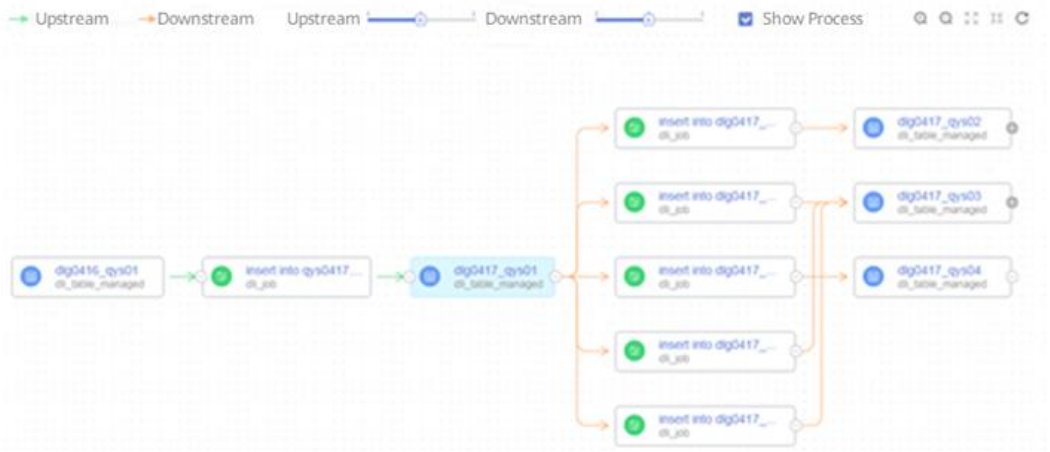


Figure 10-6 Full-link data lineages

- Data maps

  Data maps facilitate data search, analysis, development, mining, and operations. With data maps, user can search for data quickly and perform lineage and impact analysis with ease.

  Keyword search and fuzzy search are supported, helping users quickly locate the data required.

  Data maps can also be used to query table details by table names, helping users quickly master usage rules. After obtaining the detailed data information, users can add additional description.

  Data maps display the source, destination, and processing logic of each table and field.

  Users can classify and tag data assets as required.

10.2.6 DataArts DataService: Improved Access, Query, and Search Efficiency

DataArts DataService provides a unified data service bus for your enterprise. It provides centralized management of internal and external APIs; supports access, query, and retrieval of business themes, profiles, and metrics; improves data consumption experience and efficiency; and finally monetizes data assets. Users can use DataArts DataService to generate APIs and register the APIs with DataArts DataService for unified management and publication.

DataArts DataService uses a serverless architecture. Users only need to focus on the API query logic, without worrying about infrastructure such as the runtime environment. DataArts DataService supports elastic scaling of compute resources, significantly reducing O&M costs.

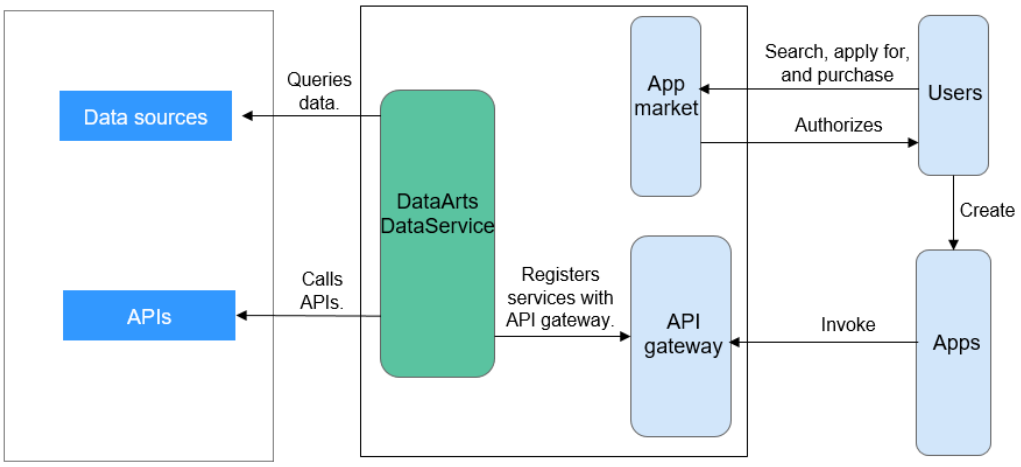


Figure 10-7 DataArts DataService architecture

10.2.7 DataArts Security: All-Round Protection

- Network security

  Tenant isolation and access permissions control are used to ensure the privacy and data security of systems and users based on network isolation and security group rules, as well as security hardening measures.

- User permissions control

  Role-based access control allows users to associate roles with permissions and supports fine-grained permission policies, meeting different authorization requirements. DataArts Studio provides four roles for different users: admin, developer, operator, and viewer. Each role has different permissions.

- Data security

  DataArts Studio provides a review mechanism for key processes.

  Data is managed by level and category throughout the lifecycle, ensuring data privacy compliance and traceability.

10.2.8 DataArts Studio Advantages

- One-stop data operations platform

  DataArts Studio is a one-stop data operations platform that allows you to perform many operations, including integrating data from every domain, designing data architecture, monitoring data quality, managing data assets centrally, developing data services, and connecting data from different data sources. In a word, it can help you build a comprehensive data governance solution.

- Comprehensive data control and governance

  DataArts Studio enables users to monitor their data quality in the full data lifecycle, provides users with standard data definitions and visualized model design, generates data processing code, and notifies users immediately when anomaly events occur.

- Diverse data development types

DataArts Studio has a wide range of scheduling configuration policies and powerful job scheduling capability. It supports online collaborative development among multiple users, online editing and real-time query of SQL and shell scripts, and job development via data processing nodes such as CDM, SQL, MRS, Shell, and Spark.

- Unified scheduling and O&M

  Fully hosted scheduling based on time and event triggering mechanisms enables task scheduling by minute, hour, day, week, or month.

  The visualized task O&M center monitors all tasks and supports alarm notification, enabling users to obtain real-time task status and ensuring normal running of services.

- Reusable industrial knowledge bases

  DataArts Studio provides vertical industries with reusable knowledge bases, including data standards, domain models, subject libraries, algorithm libraries, and metric libraries, and supports fast customization of E2E data operations solutions for industries such as smart government, smart taxation, and smart campus.

- Unified data asset management

  DataArts Studio allows users to have a global view of data assets, facilitating fast asset query, intelligent asset management, data source tracing, and data openness. In addition, it enables users to define business data catalog, terms, and classifications, as well as access to assets in a unified manner.

- Visualized data operations in all scenarios

  Visualized data governance requires only drag-and-drop operations without coding; visualized processing results facilitate interaction and exploration; visualized data asset management supports drill-down and source tracing.

- All-round security assurance

  Unified security authentication, tenant isolation, data grading and classification, and data lifecycle management ensure data privacy, auditability, and traceability. Role-based access control allows users to associate roles with permissions and supports fine-grained permission policies, meeting different authorization requirements.

10.3 Quiz

What functions does DataArts Studio provide?

11 Summary

This document introduces basic big data knowledge, including components such as HDFS, Hive, HBase, ClickHouse, MapReduce, Spark, Flink, Flume, and Kafka, how to use MRS, and MRS operations and development. It also introduces big data solutions and data mining, which will be described in detail in the latest HCIP and HCIE certification courses.

In addition, you can obtain more learning resources in the following ways:

1. Huawei Talent: https://e.huawei.com/en/talent/portal/#/
2. Huawei iLearningX platform: https://ilearningx.huawei.com/portal/
3. WeChat official accounts:
   - Huawei Certification
   - Huawei HCIE Elite Salon
   - Huawei ICT Academy