

Big Data Development Trends



Foreword

- This chapter details big data development trends, including an overview of big data, applications for big data, opportunities and challenges in the big data era.

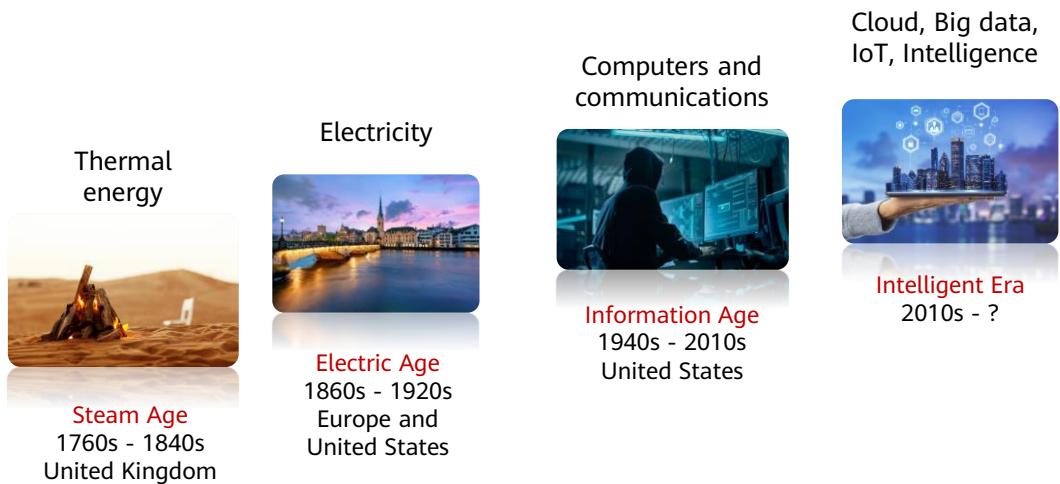
Objectives

- Upon completion of this course, you'll understand:
 - Concept and four Vs of big data
 - Trends and applications for big data
 - Opportunities and challenges in the big data era

Contents

- 1. Big Data Era**
2. Big Data Application Fields
3. Challenges and Opportunities Faced by Enterprises

Arrival of the Fourth Industrial Revolution and Intelligent Era



- In the 1760s, Britain took the lead in launching the first industrial revolution, bringing the world into the steam age. Hence, Britain became the first industrialized country in the world and brought a series of social changes.
- In the middle of the 19th century, the completion of bourgeois revolution or reforms in European countries, the United States and Japan promoted economic development. In the late 1860s, the second industrial revolution began. With the large-scale use of alternating current, the world 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 gradually led the world.
- The fourth technological revolution is on the rise. A revolution centering on emerging IT technologies such as cloud, big data, IoT, and intelligence is in full swing. Responding to the call of the times and becoming the "trend" of the times is the direction we are aiming at.

Intelligent World: From Data Management to Data Operation

Data Determines Experience



120 million data tags per day

Data determines the user experience

Data Determines Decision



Comprehensive analysis of historical oilfield data over the past 50 years

Data analysis determines oil production efficiency

Data Determines Process

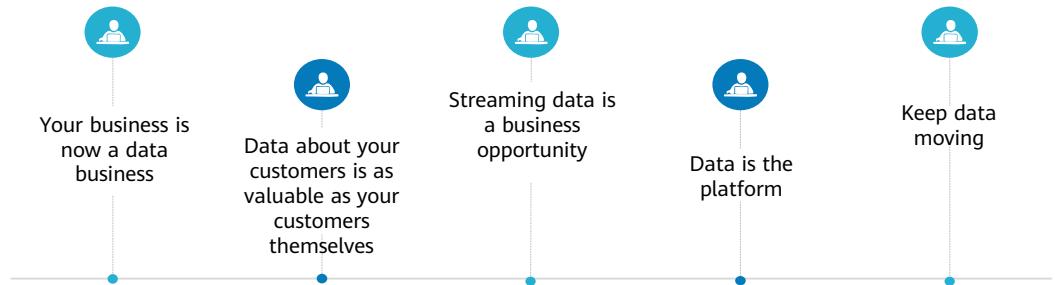


450,000 data collisions per day

Data flow streamlines processes

- In addition to national strategic requirements, operators need to transform their mindsets to adapt to the digital and information era.
- In the digital era, we need to become not only data managers, but also data operators, because data determines user experience, decision-making, and processes.

All Businesses Are Data-Related



Big Data Era

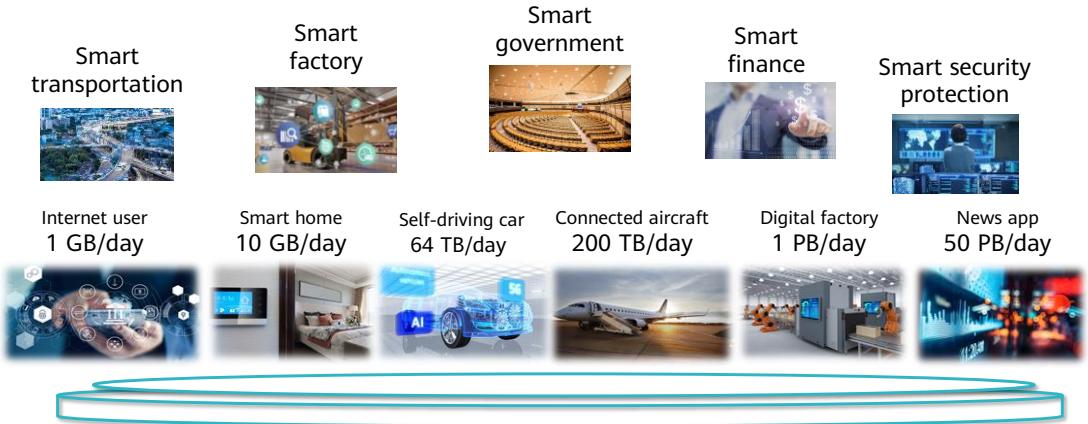
- Wikipedia:
 - Big data refers to datasets that are too large or complex to be captured, managed, and dealt with by traditional data-processing application software.



- On the basis of 3 Vs, the industry adds the low value density (Value) to form 4 Vs, while IBM adds authenticity and accuracy (Veracity) to form 4 Vs.
- Currently, there is no acknowledged definition of big data. Different definitions are all based on characteristics of big data.

Boom in Big Data

- China has the most number of netizens, and the amount of data generated in China on a daily basis is also among the most.



Differences Between Big Data and Traditional Data Processing

- From Database (DB) to Big Data (BD)
 - **Pond fishing vs. Sea fishing.** Fish here is the data to be processed.

	Big Data Processing	Traditional Data Processing
Data scale	Large (in GB, TB, or PB)	Small (in MB)
Data type	Multiple (structured, semi-structured, and unstructured data)	Single (mainly structured data)
Mode-data relationship	Modes are set after data is generated. Modes evolve as data increases.	Modes are set before data is generated (ponds are built before fish are put in).
Processing object	"Fish in the sea", through which certain "fish" are used to determine whether other kinds of "fish" exist.	"Fish in the pond"
Tool	No size fits all	One size fits all

Main Computing Modes of Big Data Applications

- Batch computing
 - Processes massive data in batches. Major technologies include MapReduce and Spark.
- Stream computing
 - Calculates and processes streaming data in real time. Major technologies include Spark, Storm, Flink, Flume, and Dstream.
- Graph computing
 - Processes large-scale graph structure data. Major technologies include GraphX, Gelly, Giraph, and PowerGraph.
- Query and analysis computing
 - Storage management and query analytics of massive data. Main technologies include Hive, Impala, Dremel, and Cassandra.

- GraphX: built-in graph computing algorithm library of Spark.
- Gelly: built-in graph computing algorithm library of Flink.
- Giraph: an open-source implementation of the paper "Pregel: A System for Large-Scale Graph Processing" released by Google in 2010. Giraph is an upper-layer application developed based on Hadoop. Its system architecture and computing model are the same as those of Pregel.
- PowerGraph: distributed and parallel computing framework. It proposes the node segmentation policy to solve the long tail phenomenon and power-law distribution problems in Natural Graph.
- Impala: a new query system developed by Cloudera. It provides SQL semantics and can be used to query PB-level big data stored in Hadoop HDFS and HBase. It is faster than Hive.
- Dremel: an interactive data analysis system developed by Google, which provides a faster speed than Hive and can be used as a supplement to Hive and MapReduce. It is used as a reference for Impala.
- Cassandra: an open-source distributed NoSQL database system. It was initially developed by Facebook to store data in simple formats such as inbox data and integrates the data models of Google BigTable and the completely distributed architecture of Amazon Dynamo.

Big Data Computing Tasks - I/O-Intensive Tasks

- Tasks involving network, drive, and memory I/O are I/O-intensive tasks.
- Characteristics: CPU usage is low, and most of the time is spent waiting for the completion of I/O operations (the I/O speed is much lower than the CPU and memory speeds).
- For I/O-intensive tasks, the more tasks, the higher the CPU efficiency, but there is also a limit. Most common tasks are I/O-intensive tasks, such as web applications.
- During the execution of I/O-intensive tasks, 99% of the time is spent on I/Os, and 1% is spent on CPUs. Therefore, improving network transmission and read/write efficiency are top priorities.

Big Data Computing Tasks - CPU-Intensive Tasks

- Feature: A large amount of computing is required, consuming CPU resources. For example, the computing capability of the CPU can be used to calculate the circumference and decode HD videos.
- Although CPU-intensive tasks can be completed by multiple tasks, the more tasks, the more time is spent task switching, and the lower the task execution efficiency of the CPU. Therefore, to make the most efficient use of the CPU, the number of CPU-intensive tasks that are performed at the same time should be equal to the number of CPU cores.
- CPU-intensive tasks mainly consume CPU resources. Therefore, code running efficiency is critical.

Big Data Computing Tasks - Data-Intensive Tasks

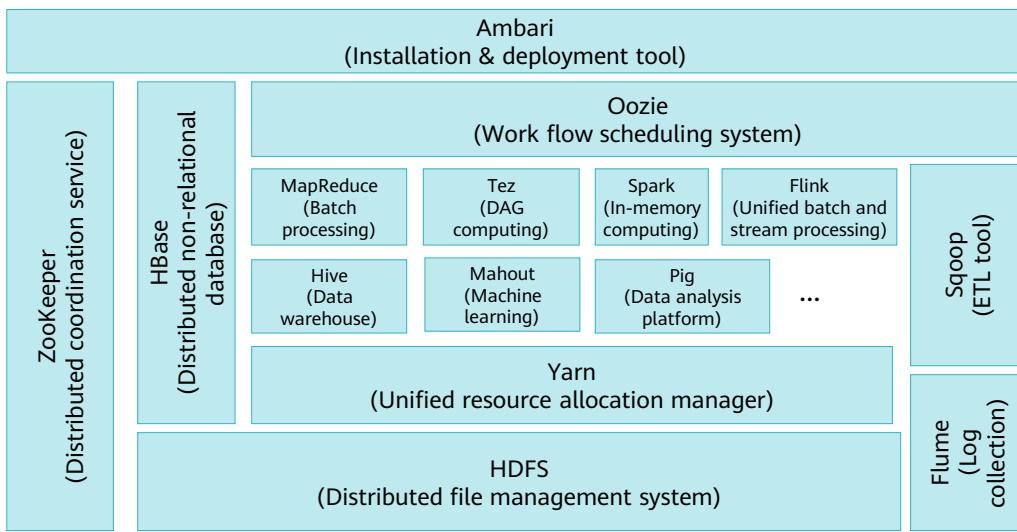
- Data-Intensive applications differ from CPU-intensive applications.
 - Traditional CPU-intensive applications tend to run a small number of computing jobs on tightly coupled supercomputers in parallel computing mode, which means that one computing job occupies a large number of compute nodes at the same time.
 - Data-intensive applications have the following characteristics:
 - A large number of independent data analysis and processing jobs can be distributed on different nodes of a loosely coupled computer cluster system.
 - Highly intensive massive data I/O requirements;
 - Most data-intensive applications have a data flow-driven process.

Big Data Computing Tasks - Data-Intensive Tasks

- Typical applications of data-intensive computing are classified into the following types:
 - Log analysis of Web applications
 - Software as a Service (SaaS) applications
 - Business intelligence applications for large enterprises

- Web applications: Both traditional search engines and emerging Web 2.0 applications are Internet service systems based on massive data and centered on data processing. To support these applications, the system needs to store, index, and back up massive heterogeneous web pages, user access logs, and user information (profiles), and ensure quick and accurate access to the data. Obviously, this requires the support of data-intensive computing systems. Therefore, web applications become the source of data-intensive computing.
- SaaS applications: SaaS provides open software service interfaces so that users can obtain customized software functions on a public platform, reducing the purchase and maintenance costs of software and hardware platforms and making it possible to integrate applications and services. Data involved in various user applications is massive, heterogeneous, and dynamic. To effectively manage and integrate the data and provide data convergence and interoperability functions while ensuring data security and privacy, the data-intensive computing system is required.
- Business intelligence applications of large enterprises: Large enterprises are usually geographically distributed across regions. The Internet provides a platform for unified management and global decision-making. To implement enterprise business intelligence, a series of subsystems such as production, sales, supply, service, personnel, and finance need to be integrated. Data is one of the objects of integration and the basis of business intelligence. The data in these systems includes product design, production process, plan, customer, order, and pre-sales and post-sales service data. In addition to various types and quantity, the structure is complex and heterogeneous. Data-intensive computing system is a supporting technology to realize business intelligence of cross-region enterprises.

Hadoop Big Data Ecosystem



Contents

1. Big Data Era
2. **Big Data Application Fields**
3. Challenges and Opportunities Faced by Enterprises

Big Data Era Leads the Way

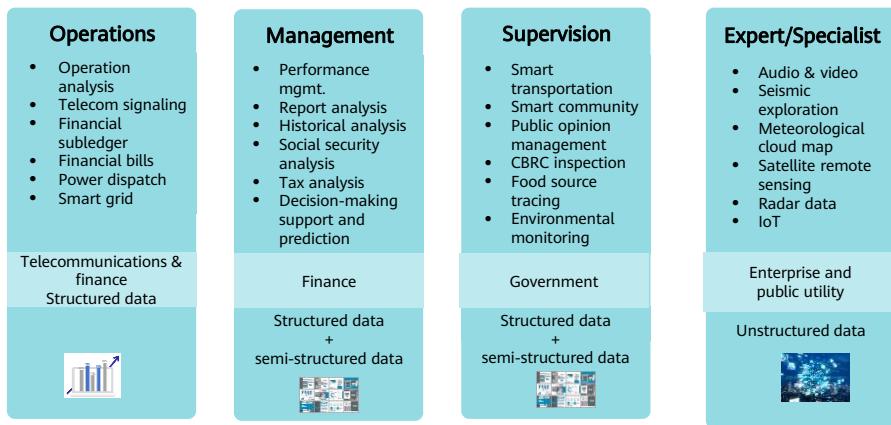
Data has penetrated every industry and business domain.

Insights into the essence (business), trend prediction, and future orientation are the key elements of the big data era. The future guides the present, and the present ensures the future.



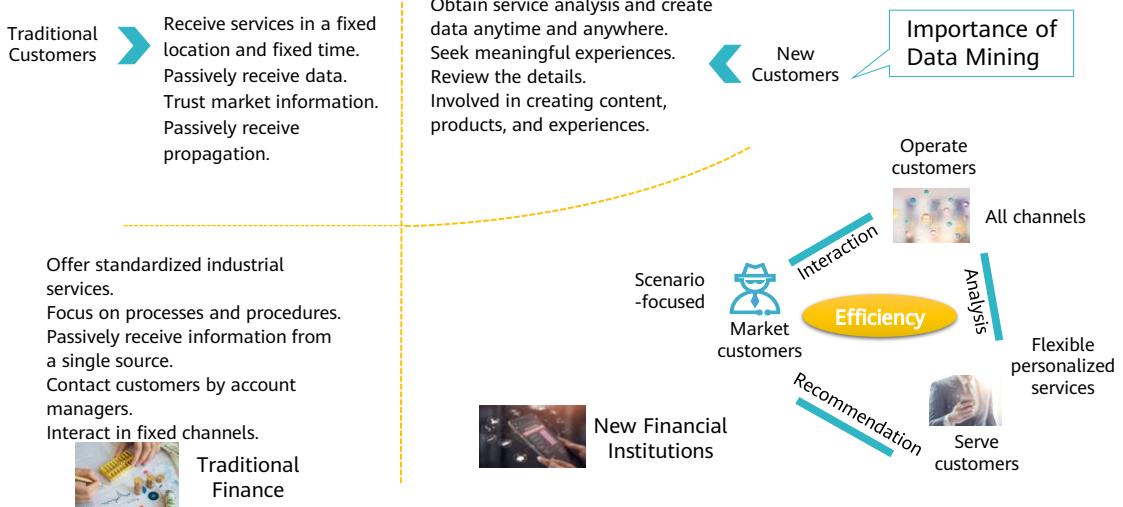
- Social networking and IoT technologies expand technology channels of data collection.
- Distributed storage and computing technologies are the technical basis of big data processing.
- Emerging technologies such as deep neural networks usher in a new era of big data analysis technologies.

Application Scenarios of Enterprise-Level Big Data Platforms



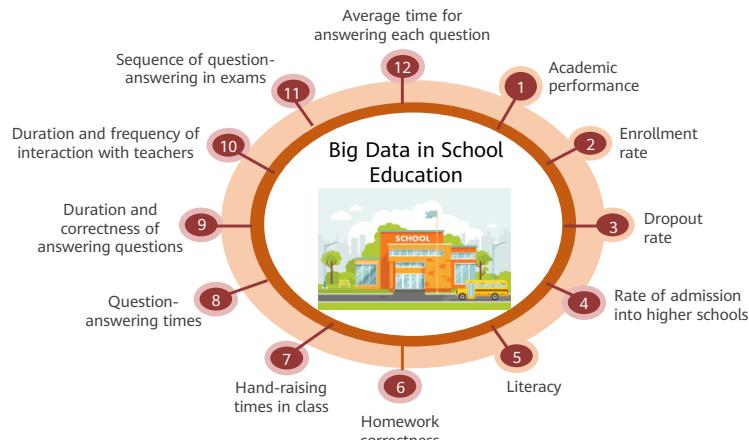
Industries such as telecom, finance, and government have strong requirements for data analysis. New Internet technologies have been adopted to process big data with low value density.

Big Data Application - Finance



Big Data Application - Education

- Big data analysis has been applied in all facets of education.



- **Education reform**

- Big data provides technical support for basic and higher education and analyzes all important information, including students' psychological activities, learning behaviors, exam scores, and career planning. Most of the 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 analysis is to improve students' learning performance and use them to provide personalized services for improving student achievement. At the same time, it can improve students' final examination results, attendance, drop-out rate, and promotion rate, and balance the education development.

- **Learning analysis**

- Nowadays, teachers no longer use big data to display only students' scores and homework results. Instead, they can get to know 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 diagnose problems in a timely manner with more detailed information, such as the time and frequency of readings, give suggestions for improvement, and predict students' final exam results.

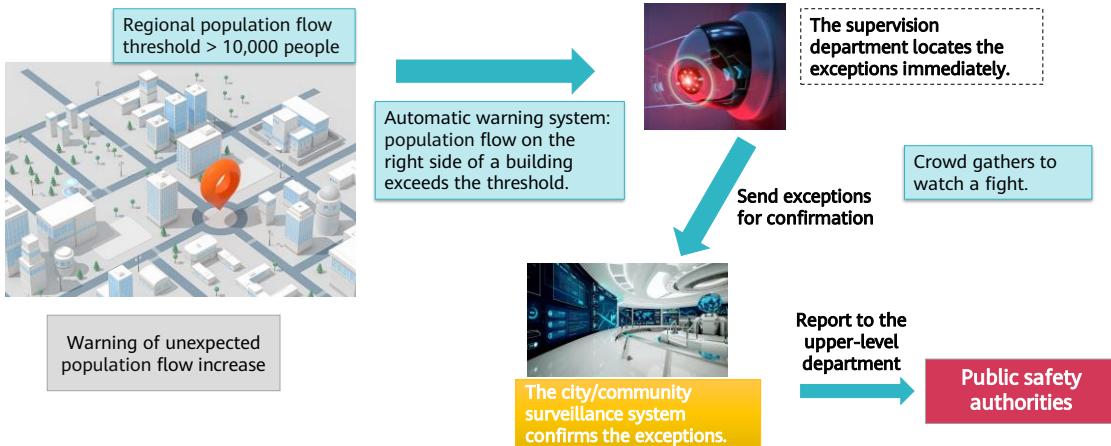
- **Exam evaluation**

- 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 is the longest time? What is the shortest time? What's the average time? For questions that have been asked before, whether the students have answered the questions correctly or incorrectly? What are the clues to the problem that benefit 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 customized

teaching in accordance with the students' aptitude.

Big Data Application - Public Security

Public Safety Scenario - Automatic Warning and Linkage



22 Huawei Confidential



- Use cloud computing and big data to locate the areas with the highest crime occurrence, and create a hotspot map of such areas based on massive data. 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 to locate the crime-prone point and catch the fugitive.

Big Data Application - Transportation Planning

Traffic Planning Scenario - Multi-dimensional Traffic Crowd Analysis

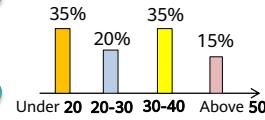


Traffic prediction suggestions based on crowds

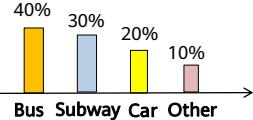
Area where historical people flows exceed the threshold

- North gate of the Beijing Workers' Sports Complex: The population flow exceeds 500/hour.
- Sanlitun: The population flow exceeds 800/hour.
- Beijing Workers' Sports Complex: 1,500 people/hour

Analysis by crowd



Analysis by travel mode proportion



Road network planning



Bus route planning



Big Data Application - Clean Energy

Clean Energy Power Supply in Qinghai for Nine Consecutive Days

Coal Consumption Reduced by
800,000 Tons

CO₂ Emissions Reduced by
1.44 Million Tons



"Intelligence + Data": Clearing the Air and Water

Contents

1. Big Data Era
2. Big Data Application Fields
3. Challenges and Opportunities Faced by Enterprises

Challenges Faced by Traditional Data Processing

Costly storage of massive data volumes

Insufficient batch processing performance

Lack of streaming data processing

Limited scalability

Single data source

External value-added data assets

Challenge 1: Indefinite Big Data Requirements

- Many enterprise business departments do not understand the application scenarios and values of big data, making it difficult to raise accurate requirements for big data. Since the requirements are not clear and the big data department is a non-profit department, 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.

Challenge 2: Data Silos

- Data fragmentation is the most important challenge for enterprises when implementing big data.
 - In a large enterprise, different types of data are scattered in 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.

- Big data requires association and integration of different data to better leverage the advantages of understanding customers and services.

Challenge 3: Low Data Availability and Poor Quality

- Many large- and medium-sized enterprises generate a large amount of data on a daily basis, 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 denoising. Sybase data shows that if the availability of high-quality data is improved by 10%, enterprise profits can increase by more than 20%.

Challenge 4: Data-related Management Technologies and Architectures

- Traditional databases are not suitable for processing PB-level data.
- Traditional databases do not support data diversity, especially compatibility with structured, semi-structured and unstructured data.
- Massive data O&M needs to ensure data stability, support high concurrency, and reduce server loads.

Challenge 5: Data Security Issues

- Networked life makes it easier for criminals to obtain information about people, and has given rise to new forms of crime that are difficult to track and prevent.
- Ensuring user information security has become an important issue in the big data era. 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 copies and disaster recovery mechanisms.

Challenge 6: Shortage of Big Data Talent

- Each phase of big data construction depends on professional personnel. Therefore, a professional big data construction team with management skills and big data application experience must be cultivated. 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 one million. Therefore, universities and enterprises must work together to find and cultivate talent.

Challenge 7: Trade-off Between Data Openness and Privacy

- As big data applications become increasingly important, data resource openness and sharing have become key to maintaining advantages in the data war. However, data openness will inevitably infringe on some users' privacy. Protecting citizens' and enterprises' privacy while promoting data openness, application, and sharing, and strengthening privacy legislation are key challenges in the big data era.

Big Data Blue Ocean as a New Focus of Enterprise Competition

- The enormous business value of big data is believed to be equivalent to the computer revolution of the 20th century. Big data affects every field, including business and the overall economy, and has led to the emergence of new blue oceans and economic growth points, becoming a new focus of enterprise competition.

- 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, online network transmission and interactive social networks made people who used to be just web-browsers become consumers. 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 become a hotly contested spot.

Opportunity 1: Big Data Mining as a New Core of Business Analysis

- The focus of big data has gradually shifted from storage and transmission to data mining and application, which will profoundly affect enterprise business models. Big data can bring direct profits to enterprises and the positive feedback provides competitive advantages that are difficult to replicate.
 - Big data technologies can help enterprises integrate, mine, and analyze the huge amount of data at their disposal, build a systematic data system, and improve their own structures and management mechanisms.
 - With the growth of consumers' personalized requirements, big data has been applied across a wide range of fields, altering the development path and business models of most enterprises.

Opportunity 2: Big Data Provides Support for Information Technology Applications

- Big data processing and analysis have become the support for next-generation information technology applications.
 - Next-generation information technologies, such as the mobile Internet, IoT, social networking, digital home, 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 a wide range of applications to further improve the user experience and create new business, economic, and social value.
 - Therefore, big data has the potential to accelerate social transformations. However, to unleash this energy, we need strict data governance, insightful data analysis, and an environment that stimulates management innovation.

Opportunity 3: Big Data as a New Engine for Growth in the Information Industry

- The business value and market requirements of big data have become a new engine that drives growth in the information industry.
 - As the value recognition of big data increases, the market demand will surge, and new technologies, products, services, and business forms oriented to the big data market will emerge on an ongoing basis.
 - 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 an important impact on the chip and storage industry. In addition, the demands for integrated data storage and processing servers and in-memory computing 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 of the data mining and business intelligence markets.

Quiz

1. (Multiple-answer question) Which of the following statements about big data features are correct? ()
 - A. Large data volume
 - B. Wide range of data types
 - C. High-speed processing
 - D. High value density
2. What challenges do we face in the big data era?

- Answer:
 1. ABC
 2. The challenges are:
 - Business departments do not have clear big data requirements.
 - Severe internal data silos.
 - Low data availability and quality.
 - Data-related management technologies and architectures.
 - Data security.
 - Lack of big data talents.
 - Trade-off between data openness and privacy.

Summary

- This chapter provides an overview of big data, including the definition of big data, four Vs of big data, application fields of big data, and challenges and opportunities in the big data era.

Acronyms and Abbreviations

- SaaS: software as a service
- IoT: Internet of Things
- IaaS: infrastructure as a service
- ECS: Elastic Cloud Server
- BMS: Bare Metal Server
- RDS: Relational Database Service
- DWS: Data Warehouse Service
- SSD: Solid State Disk

Acronyms and Abbreviations

- IOPS: input/output operations per second, which is used to measure the performance of computer storage devices.
- ROCE: RDMA over Converged Ethernet, which is a network protocol that allows applications to implement remote memory access over the Ethernet.
- HPC: high-performance computing
- EVS: Elastic Volume Service
- OBS: Object Storage Service
- SFS: Scalable File Service
- CCE: Cloud Container Engine

Acronyms and Abbreviations

- CCI: Cloud Container Instance
- VPC: Virtual Private Cloud
- ELB: Elastic Load Balance
- DLV: Data Lake Visualization
- DWS: Data Warehouse Service
- CSS: Cloud Search Service
- GES: Graph Engine Service

Recommendations

- HUAWEI CLOUD official link:
 - <https://www.huaweicloud.com/>
- MRS user guide:
 - <https://support.huaweicloud.com/intl/en-us/mrs/index.html>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



HDFS — Hadoop Distributed File System & ZooKeeper



Foreword

- This chapter describes the big data distributed storage system HDFS and the ZooKeeper distributed service framework that resolves some frequently-encountered data management problems in distributed applications. This chapter lays a solid foundation for subsequent component learning.

Objectives

- Upon completion of this course, you will be able to:
 - Understand what HDFS is and its use cases.
 - Be familiar with HDFS-related concepts.
 - Master the system architecture of HDFS.
 - Master the key features and data read/write process of HDFS.
 - Understand ZooKeeper-related concepts.
 - Master the usage of ZooKeeper.

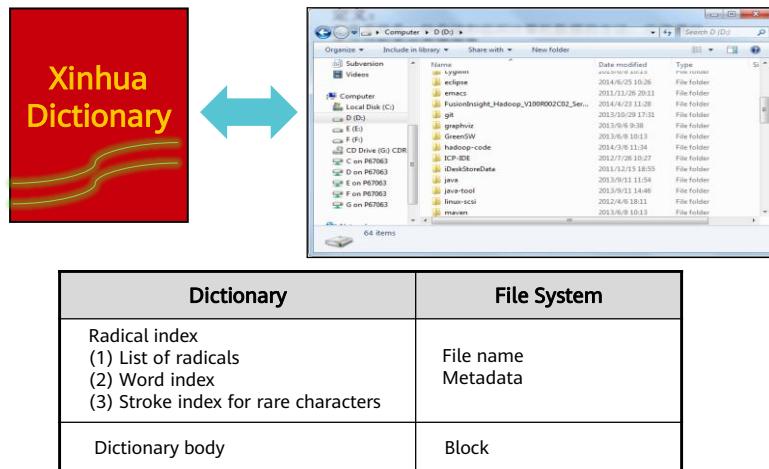
Contents

1. HDFS

- HDFS Overview
- HDFS-related Concepts
- HDFS Architecture
- HDFS Key Features
- HDFS Data Read/Write Process

2. ZooKeeper Distributed Coordination Service

Dictionary and File System



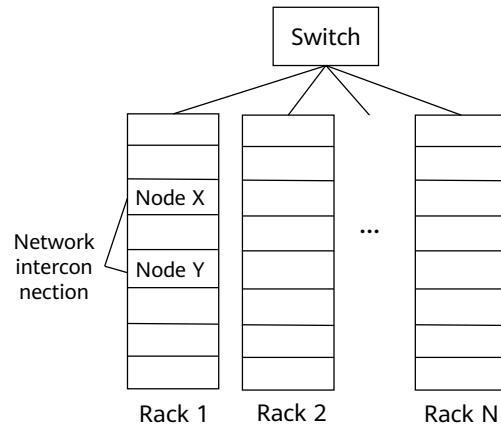
- File system: A file system is a method of storing and organizing computer data. It makes it easier to access and search for computer data.
- File name: File names in a file system are used to locate the location where files are stored.
- Metadata: File attribute data, such as the file name, file length, user group to which the file belongs, and file storage location.
- Block: A block is the minimum unit for storing files. It divides a storage medium into fixed areas, which are allocated and used as required.

Distributed File System

In a distributed file system, files are stored on multiple computer nodes.

Thousands of computer nodes form a computer cluster.

The computer nodes used in a distributed file system are general-purpose hardware, which means that hardware overheads can be significantly reduced.



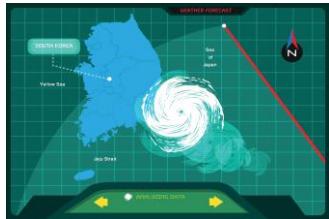
- Previously, parallel processing devices using multiple processors and dedicated advanced hardware were used.
- Physically, a distributed file system consists of multiple nodes in a computer cluster. These nodes are classified into two types: master nodes and slave nodes.

HDFS Overview

- Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
- HDFS was originally built as the infrastructure for the Apache Nutch web crawler software project.
- HDFS is part of the Apache Hadoop Core project.
- HDFS is highly fault-tolerant and is deployed on cost-effective hardware.
- HDFS provides high-throughput access to application data and is suitable for applications with large data sets.
- HDFS relaxes Portable Operating System Interface (POSIX) requirements to enable streaming access to file system data.

- Inapplicable scenarios:
 - [1] Applications with low-latency data access, for example, within dozens of milliseconds.
 - Cause: HDFS is optimized for applications with high data throughput, which causes high latency.
 - [2] A large number of small files.
 - Cause: When started, NameNode stores the metadata of the file system in the memory. Therefore, the total number of files that can be stored in the file system is restricted by the NameNode memory. Based on experience, the storage information of each file, directory, and block occupies about 150 bytes. If there are 1 million files and each file occupies one block, at least 300 MB memory is required. If you store 1 billion files, a very large amount of memory is required.
 - [3] Multi-user random write and modification of files.
 - Cause: HDFS files now have only one writer, and write operations are always written at the end of the files. In addition, the files cannot be modified at any position. This scenario may be supported in the future, but its efficiency is low.

Use Cases of HDFS



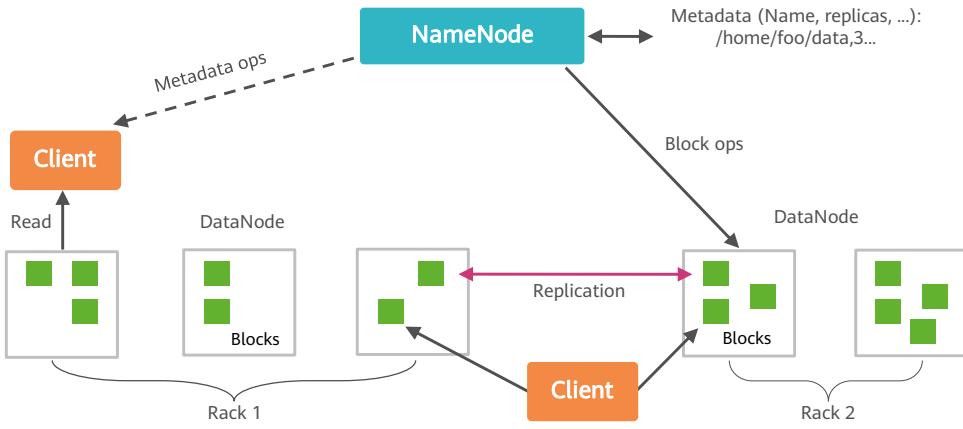
Contents

1. HDFS

- HDFS Overview
- HDFS-related Concepts
- HDFS Architecture
- HDFS Key Features
- HDFS Data Read/Write Process

2. ZooKeeper Distributed Coordination Service

Basic HDFS System Architecture



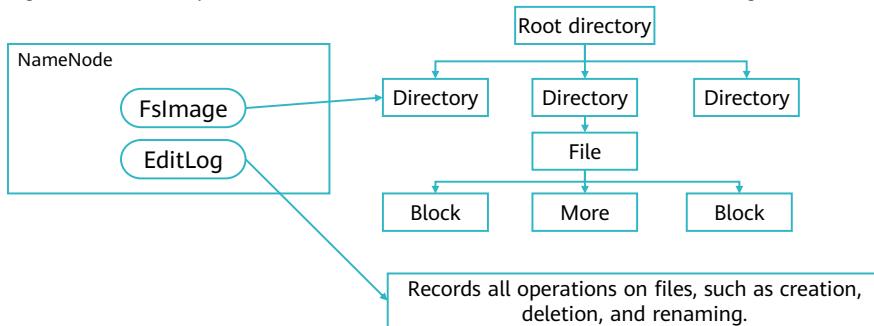
- The HDFS architecture consists of three parts: NameNode, DataNode, and Client.
 - NameNode: stores and generates metadata of file systems. It runs one instance.
 - DataNode: stores the actual data and reports blocks it manages to the NameNode. A DataNode runs multiple instances.
 - Client: supports service access to HDFS. It obtains data from the NameNode and DataNode and sends to services. Multiple instances run together with services.

Client

- Clients are the most common way of using HDFS. HDFS provides a client during deployment.
- It is a library that contains HDFS interfaces that hide most of the complexity in HDFS implementation.
- Strictly speaking, it is not part of HDFS.
- It supports common operations such as opening, reading, and writing, and provides a shell-like command line mode to access data in HDFS.
- HDFS also provides Java APIs that serve as client programming interfaces for applications to access the file system.

NameNode

- In HDFS, the NameNode manages the namespace of the distributed file system and stores two core data structures: FsImage and EditLog.
 - FsImage maintains the metadata of the file system tree and all the files and folders in that file tree.
 - EditLog records all the operations on files, such as creation, deletion, and renaming.



- The NameNode records the location information of the DataNode; where each block in each file is located.

DataNode

- A DataNode is a worker node of HDFS. It stores and retrieves data based on the scheduling of Clients or NameNodes, and periodically sends the list of blocks stored on it to NameNodes.
- Data on each DataNode is stored in the local Linux file system of the node.

NameNodes vs. DataNodes

NameNode	DataNode
Stores metadata.	Stores file content.
Stores metadata in the memory.	Stores file content in the disk.
Stores the mapping between files, blocks, and DataNodes.	Maintains the mapping between block IDs and local files on DataNodes.

Block

- The default size of a block is 128 MB. A file is divided into multiple blocks. A block is the storage unit.
- The block size is much larger than that of a common file system, minimizing the addressing overhead.
- A block has the following benefits:
 - Large-scale file storage
 - Simplified system design
 - Data backup

- Support large-scale file storage: A file is stored in blocks. A large file can be split into multiple file blocks, which are distributed to different nodes. Therefore, the size of a file is not constrained by the storage capacity of a single node.
- Streamline system design: Firstly, storage management is greatly simplified since the file block size is fixed and it is easy to calculate the number of file blocks that can be stored on a node. Secondly, metadata can be managed by other systems instead of file blocks.
- Facilitate data backups: Each file block can be redundantly stored on multiple nodes, greatly improving system fault tolerance and availability.

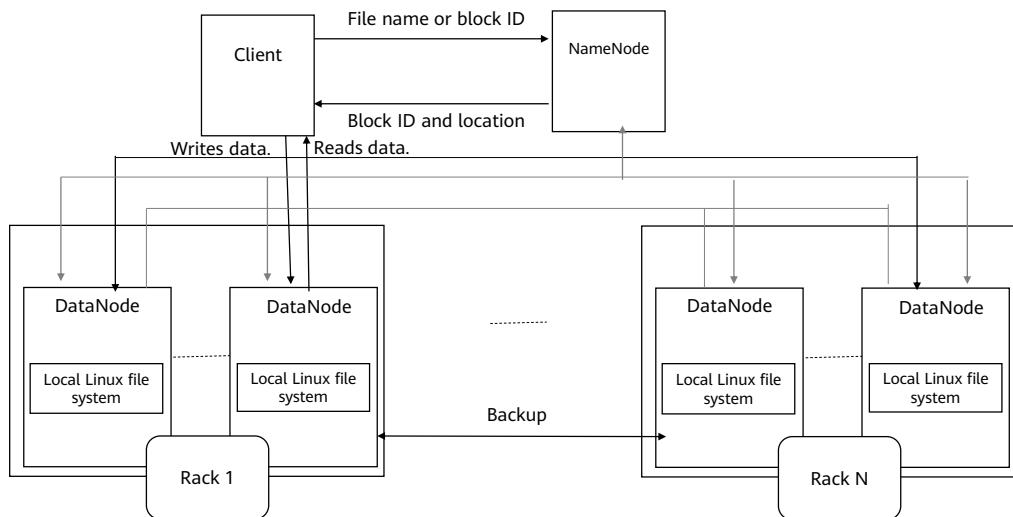
Contents

1. HDFS

- HDFS Overview
- HDFS-related Concepts
- **HDFS Architecture**
- HDFS Key Features
- HDFS Data Read/Write Process

2. ZooKeeper Distributed Coordination Service

HDFS Architecture



- HDFS uses the master/slave structure. An HDFS cluster consists of one NameNode and several DataNodes.
 - The NameNode is the centerpiece of the HDFS and manages the namespace of the file system and client access to files.
 - A DataNode runs a process in the cluster. DataNodes process read and write requests from file system clients and create, delete, and replicate data blocks under the unified scheduling of NameNodes.
 - Data on each DataNode is stored in the local Linux file system.

HDFS Namespace Management

- An HDFS namespace contains directories, files, and blocks.
- HDFS uses the traditional hierarchical file system. Therefore, users can create and delete directories and files, move files between directories, and rename files in the same way as with common file systems.
- NameNodes maintain file system namespaces. Any changes to the file system namespace or its attributes are recorded by the NameNode.

- NameNodes maintain file system namespaces. Any changes to the file system namespace or its attributes are recorded by the NameNode. The application can specify the number of file replicas that should be maintained by HDFS. The number of copies of a file is called the replication factor of the file. This information is stored by the NameNode.

Communication Protocol

- HDFS is a distributed file system deployed on a cluster. Therefore, a large amount of data needs to be transmitted over the network.
 - All HDFS communication protocols are based on TCP/IP.
 - The Client initiates a TCP connection to the NameNodes through a configurable port and uses the Client protocol to interact with the NameNodes.
 - The NameNode and the DataNodes interact with each other through the DataNode protocol.
 - Interaction between the Client and the DataNodes is implemented through the Remote Procedure Call (RPC). In design, the NameNode does not initiate an RPC request, but responds to RPC requests from the Client and DataNodes.

Disadvantages of the HDFS Single-NameNode Architecture

- Only one NameNode is configured for HDFS, which significantly simplifies the system's design but also has the following disadvantages:
 - **Namespace limitation:** NameNodes are stored in the memory. Therefore, the number of objects (files and blocks) that can be contained in a NameNode is limited by memory size.
 - **Performance bottleneck:** The throughput of the entire distributed file system is limited by the throughput of a single NameNode.
 - **Isolation:** Because there is only one NameNode and one namespace in the cluster, different applications cannot be isolated.
 - **Cluster availability:** Once the only NameNode is faulty, the entire cluster becomes unavailable.

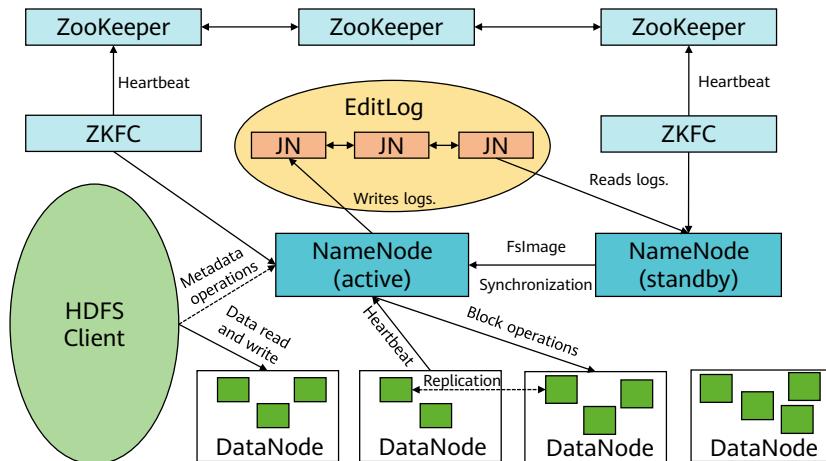
Contents

1. HDFS

- HDFS Overview
- HDFS-related Concepts
- HDFS Architecture
- **HDFS Key Features**
- HDFS Data Read/Write Process

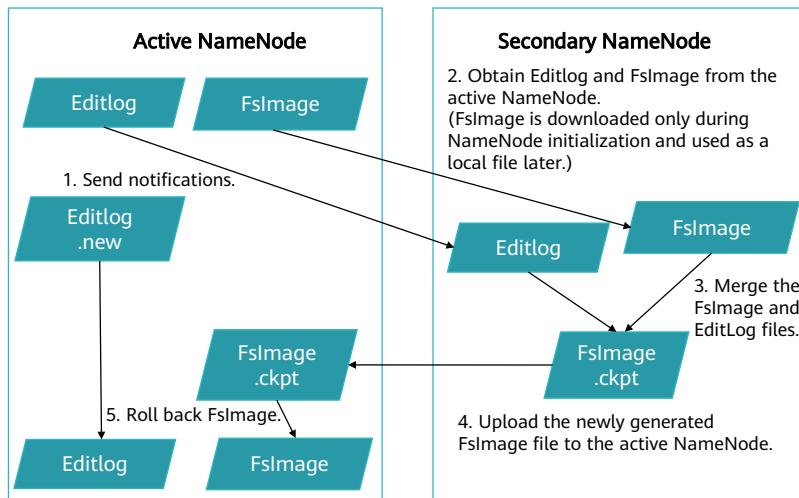
2. ZooKeeper Distributed Coordination Service

HDFS High Availability



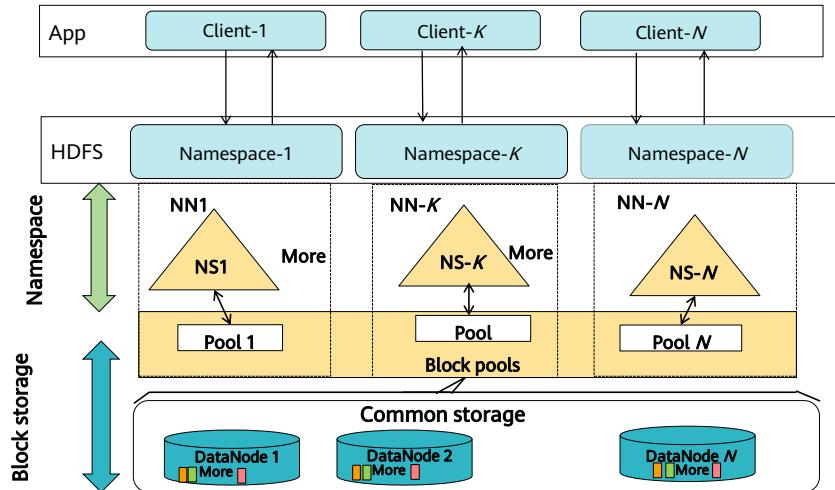
- ZooKeeper is used to implement active and standby NameNodes to eliminate single points of failure (SPOFs). This ensures the high reliability of HDFS.
- ZooKeeper is used to store HA status files and active/standby information. It is recommended that the number of ZooKeepers be an odd number greater than or equal to 3.
- NameNodes work in active/standby mode. The active NameNode provides services, and the standby NameNode synchronizes metadata from the active NameNode and functions as the hot backup of the active NameNode.
- ZooKeeper Failover Controller (ZKFC) is used to monitor the active/standby status of NameNodes.
- JournalNodes (JNs) store the Editlog generated by the active NameNode. The standby NameNode loads the Editlog on the JNs and synchronizes metadata.
- ZKFC arbitrates which NameNode is the active node and which NameNode is the standby node.
 - As a simple arbitration agent, ZKFC uses the distributed lock function of ZooKeeper to implement active/standby arbitration and controls the active/standby status of NameNodes through the command channel. ZKFC and NameNodes are deployed together, and their numbers should be the same.
- Metadata synchronization
 - The active NameNode provides services for external systems. The generated Editlog is written to the local host and JN at the same time, and updates the metadata in the memory of the active NameNode.
 - When detecting changes to the Editlog on the JN, the standby NameNode loads the Editlog to the memory and generates new metadata the same as that on the active NameNode. Metadata synchronization is then complete.
 - The active and standby FslImages are still stored in their respective disks and do not interact with each other. FslImage is a copy of metadata periodically written from the memory to the local disk. It is also called metadata mirroring.

Metadata Persistence



- **EditLog:** records user operations and is used to generate a new file system image based on **FsImage**.
- **FsImage:** saves file images periodically.
- **FsImage.ckpt:** merges **FsImage** and **EditLog** files in the memory, generates a new **FsImage** and writes it to the disk. This process is called a checkpoint. After loading the **FsImage** and **EditLog** files, the standby NameNode writes the merged result to the local disk and NFS. At this time, there is an original **FsImage** file and a new checkpoint file **fsimage.ckpt** on the disk. Then change the name of **fsimage.ckpt** to **fsimage** to overwrite the original **FsImage**.
- **EditLog.new:** The NameNode triggers log merging every hour or when the **Editlog** file reaches 64 MB. When data is transferred to the standby NameNode, the data cannot be read or written at the same time. In this case, the NameNode generates a new log file **Editlog.new** to store operation logs generated in this period. The standby NameNode merges the files and sends them to the active NameNode to replace the original **fsimage**. Rename **Editlog.new** to **Editlog**.

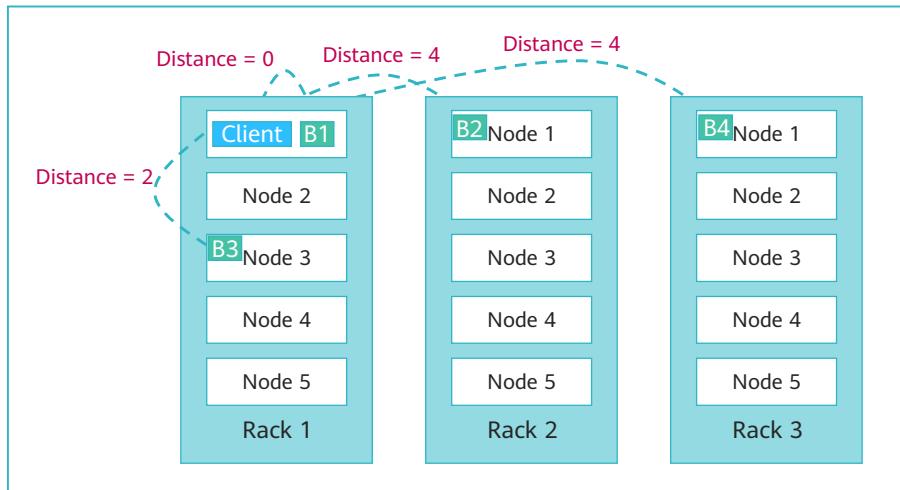
HDFS Federation



- Background: The single-active NameNode architecture cannot ensure the scalability and performance of HDFS clusters. When such a cluster expands to a certain size, the memory used by the NameNode process may reach hundreds of GB. In this case, the NameNode becomes the performance bottleneck.
- Use case: ultra-large file storage. For example, Internet companies store large-scale data such as user behavior data, historical telecom data, and voice data. In this case, the NameNode memory is insufficient to support such a large cluster.
- The common estimation formula is as follows: 1 GB corresponds to 1 million blocks. Based on the default block size, the size is approximately 128 TB. (Since this ratio is an estimation, even if each file has only one block, all metadata information does not reach 1 KB/block.)
- Federation: each NameNode is responsible for its own directory. Similar to mounting disks to directories in Linux, each NameNode is responsible for only some directories in the entire HDFS cluster. For example, if NameNode 1 is responsible for the **/database** directory, it is responsible for the metadata of all files in this directory. Metadata is not shared among NameNodes. Each NameNode has a standby node.
- Block pool: a group of file blocks that belong to a namespace.
- In the federation environment, each NameNode maintains a namespace volume, including the metadata of the namespace and the block pool of all file blocks in the namespace.
- NameNodes are independent of and do not communicate with each other. This ensures that the failure of one NameNode does not affect other NameNodes.
- DataNodes register with all NameNodes in the cluster to store data for all block pools in the cluster.
- NS: namespace. An HDFS namespace contains directories, files, and blocks. It can be

seen as the logical directory to which the NameNode belongs.

Data Replica Mechanism



- Formulas for calculating the replica distance are as follows:
 - Distance (Rack 1/D1, Rack 1/D1) = 0,
 - The distance within the same server is 0.
 - Distance (Rack 1/D1, Rack 1/D3) = 2,
 - The distance between different servers of the same rack is 2.
 - Distance (Rack 1/D1, Rack 2/D1) = 4,
 - The distance between servers of different racks is 4.
 - The distance between nodes in different data centers is 6.
- Replica placement policy:
 - The first replica is placed on the DataNode to which the file is uploaded. If the replica is submitted outside the cluster, a node whose disk space is not full and the CPU is not busy is randomly selected.
 - The second replica is placed on a node in a different rack from the first replica.
 - The third replica is placed on other nodes in the same rack as the first replica.
 - Other replicas are randomly placed on nodes.
- If the host of the write requester is a DataNode, the replica is stored locally. Otherwise, a DataNode is randomly selected from the cluster.
- Rack 1: Rack 1. D1: DataNode 1. B1: block 1 on the node.

HDFS Data Integrity Assurance

- HDFS ensures component reliability and the integrity of stored data.
- Rebuilding the replica data of failed data disks:
 - When the DataNode fails to report data to the NameNode, the NameNode initiates replica rebuilding to restore lost replicas.
- Cluster data balancing:
 - HDFS incorporates the data balancing mechanism, which ensures that data is evenly distributed on DataNodes.
- Metadata reliability:
 - The log mechanism records operations on metadata, and metadata is stored on the active and standby NameNodes.
 - The snapshot mechanism of file systems ensures that data can be restored promptly in the case of misoperations.
- Security mode:
 - HDFS provides a unique security mode to prevent faults from spreading when DataNodes or disks are faulty.

- Rebuilding the replica data of failed data disks:
 - The DataNode and NameNode periodically report data status through heartbeat messages. The NameNode manages whether blocks are completely reported. If the DataNode does not report blocks because the hard disk is damaged, the NameNode initiates replica rebuilding to restore the lost replicas.
- The security mode prevents faults from spreading.
 - When a hard disk of a node is faulty, the node enters security mode. In this mode, HDFS only supports 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 security mode.

Other Key Design Points of the HDFS Architecture

- Space reclamation mechanism:
 - HDFS supports the recycle bin mechanism and dynamic setting of replicas.
- Data organization:
 - Data is stored by block in HDFS of the operating system.
- Access mode:
 - HDFS allows data access through Java APIs, HTTP, or Shell scripts.

- Disk usage: For example, if the disk size is 100 GB and 30 GB is used, the disk usage is 30%.
- Load balancing prevents hot node issues caused by uneven data distribution among nodes.

Common Shell Commands

Type	Command	Description
dfs	-cat	Displays file content.
	-ls	Displays the directory list.
	-rm	Deletes a file.
	-put	Uploads directories or files to HDFS.
	-get	Downloads directories or files to the local host from HDFS.
	-mkdir	Creates a directory.
	-chmod/-chown	Changes the group of a file.

dfsadmin	-safemode	Performs operations in security mode.
	-report	Reports service status.

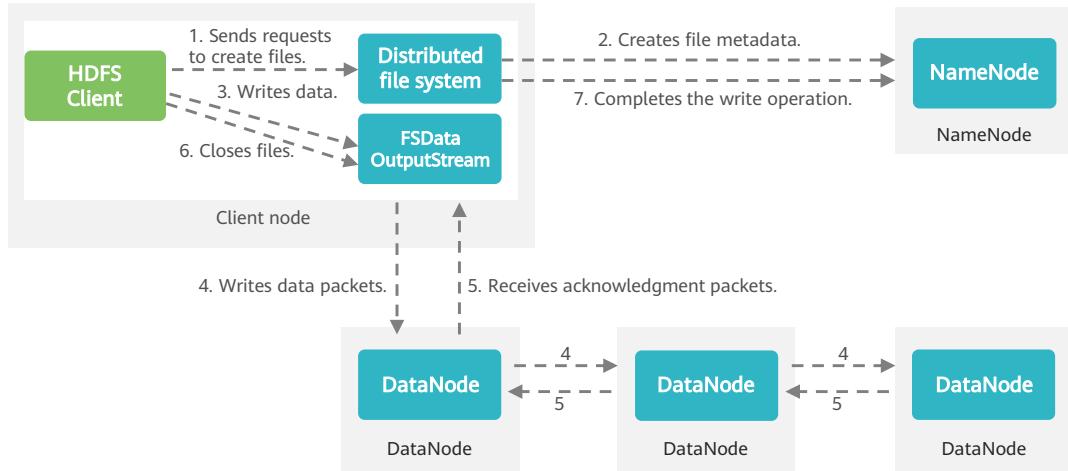
Contents

1. HDFS

- HDFS Overview
- HDFS-related Concepts
- HDFS Architecture
- HDFS Key Features
- HDFS Data Read/Write Process

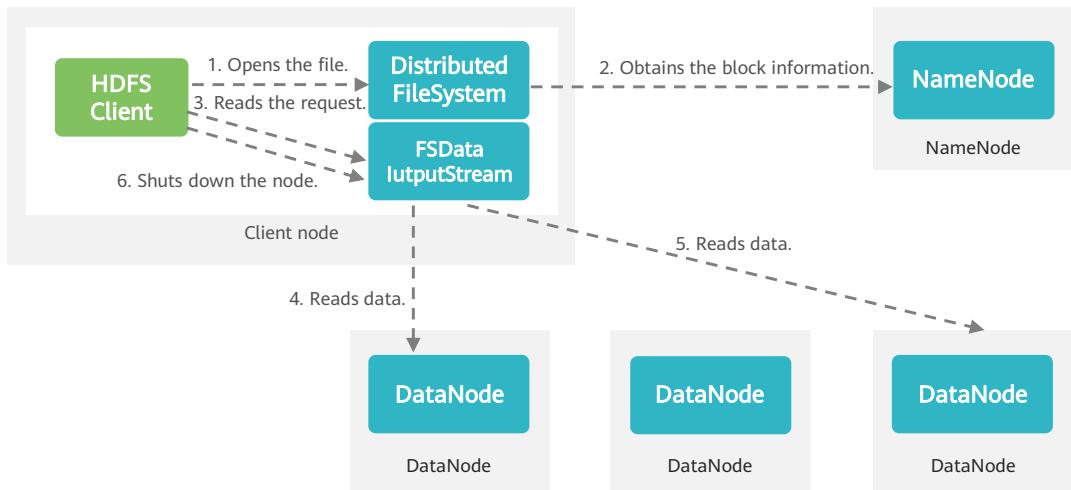
2. ZooKeeper Distributed Coordination Service

HDFS Data Write Process



- The HDFS data writing process is as follows:
 - The service application calls the HDFS Client API and requests to write data to the file.
 - The HDFS Client contacts the NameNode, which creates a file node in the metadata.
 - The service application calls the write API to write data to the file.
 - After receiving the service data, the HDFS Client obtains the data block number and location information from the NameNode, contacts the DataNode, and establishes a pipeline for DataNodes to which data is written. Then, the client writes data to DataNode1 through its own protocol, and DataNode1 replicates the data to DataNode2 and DataNode3.
 - After data write is complete, a message for confirmation is returned to the HDFS Client.
 - After all data is confirmed, the service invokes the HDFS Client to close the file.
 - After the service invokes the close and flush functions, the HDFS Client contacts the NameNode to confirm that the data write is complete. The NameNode persists the metadata.

HDFS Data Read Process



- The HDFS data read process is as follows:
 - The service application calls the HDFS Client API to open the file.
 - The HDFS Client obtains file information (blocks and DataNode location information) from the NameNode.
 - The service application calls the read API to read the file.
 - Based on the information obtained from the NameNode, the HDFS Client contacts the DataNode to obtain the corresponding blocks. (The Client reads data based on the proximity principle.)
 - The HDFS Client communicates with multiple DataNodes to obtain blocks.
 - After the data read is complete, the service invokes the close function to close the connection.

Contents

1. HDFS
2. ZooKeeper Distributed Coordination Service
 - ZooKeeper Overview
 - ZooKeeper Architecture

ZooKeeper Overview

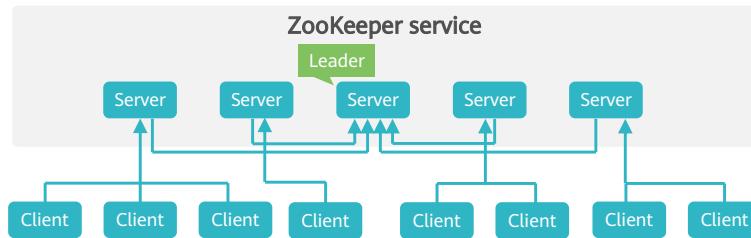
- The ZooKeeper distributed service framework is used to solve some data management problems that are frequently encountered in distributed applications and provide distributed and highly available coordination service capabilities.
- As an underlying component, ZooKeeper is widely used and depended upon by upper-layer components, such as Kafka, HDFS, HBase, and Storm. It provides functions such as configuration management, naming service, distributed lock, and cluster management.
- 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.

Contents

1. HDFS
2. **ZooKeeper Distributed Coordination Service**
 - ZooKeeper Overview
 - ZooKeeper Architecture

ZooKeeper Architecture

- A ZooKeeper cluster consists of a group of servers. In this group, there is only one leader node, with the other nodes being followers.
- The leader is elected during startup.
- ZooKeeper uses the custom atomic message protocol to ensure data consistency among nodes in the entire system.
- After receiving a data change request, the leader node writes data to the disk and then to the memory.



- A ZooKeeper cluster consists of a group of servers. In this group, one node is the leader and the other nodes are followers. When a client is connected to the ZooKeeper cluster and initiates a write request, the request is sent to the leader. Then data changes on the leader node are synchronized to the followers in the cluster.
- After receiving a data change request, the leader node writes the change to local disks for restoration. All data changes are applied to memory only after being persisted to disks.
- ZooKeeper uses ZooKeeper Atomic Broadcast (ZAB), a custom atomic protocol, to ensure the data or status consistency among nodes in the coordination system. Followers use ZAB to ensure that the local ZooKeeper data is synchronized with the leader node and provide services independently based on the local storage.
- If the leader node fails, the message layer elects another leader to act as the center of the coordination service cluster. In this way, the new leader node can process the write requests of clients and synchronize (broadcast) data changes to the ZooKeeper coordination system to other followers.

ZooKeeper Disaster Recovery

- ZooKeeper can provide services for external systems after the election is complete.
 - During ZooKeeper election, if an instance obtains more than half of the votes, the instance becomes the leader.
- For a service with N instances, N may be an odd or even number.
 - When N is an odd number, assume that $n = 2x + 1$, the node needs to obtain $x+1$ votes to become the leader, and the DR capability is x .
 - When N is an even number, assume that $n = 2x + 2$, the node needs to obtain $x+2$ (more than half) votes to become the leader, and the DR capability is x .

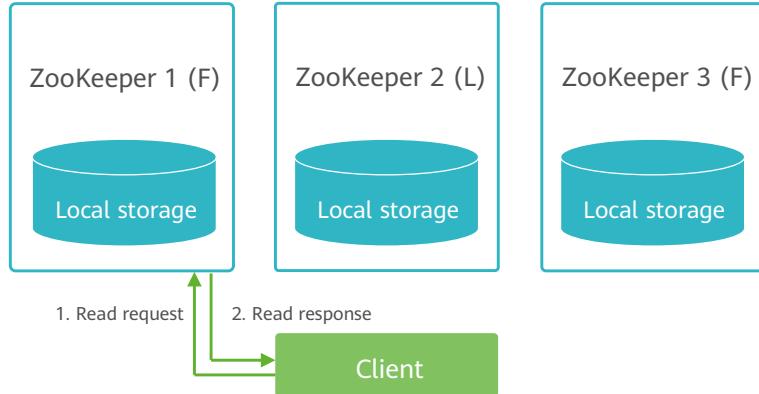
- Therefore, the DR capability of $2x+1$ nodes is the same as that of $2x+2$ nodes (three nodes are the same as four nodes, and five nodes are the same as six nodes). Considering the correlation between the election and write operation speed and the number of nodes, it is recommended that an odd number of nodes be deployed for ZooKeeper.

Key Features of ZooKeeper

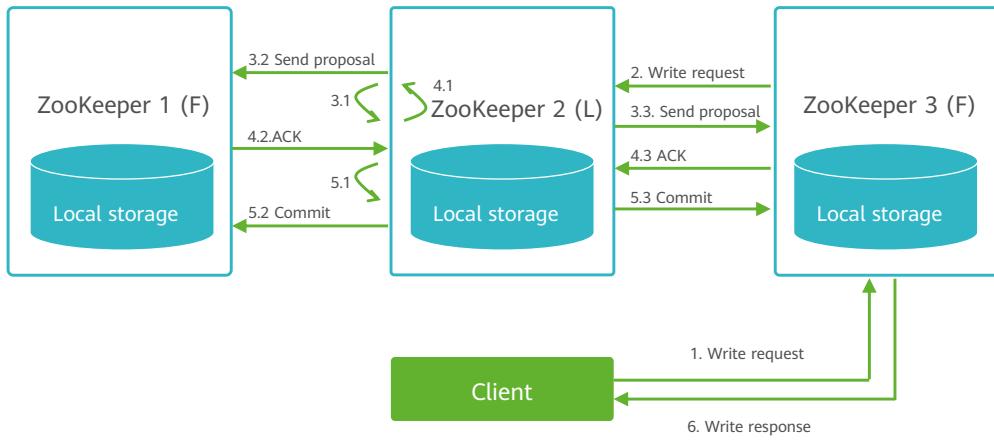
- Eventual consistency: All servers are displayed in the same view.
- Real-time: Clients can obtain server updates and failures within a specified period of time.
- Reliability: A message will be received by all servers.
- Wait-free: Slow or faulty clients cannot intervene in the requests of rapid clients so that the requests of each client can be processed effectively.
- Atomicity: Data transfer either succeeds or fails, but no transaction is partial.
- Sequence consistency: Updates sent by the client are applied in the sequence in which they are sent.

Read Function of ZooKeeper

- According to the consistency of ZooKeeper, the client obtains the same view regardless of the server connected to the client. Therefore, read operations can be performed between the client and any node.



Write Function of ZooKeeper



- The write process in the slide shows:
 - A client can initiate 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.

Common Commands for the ZooKeeper Client

- Invoking a ZooKeeper client:

```
zkCli.sh -server 172.16.0.1:24002
```

- Creating a node: **create /node**
- Listing subnodes: **ls /node**
- Creating node data: **set /node data**
- Obtaining node data: **get /node**
- Deleting a node: **delete /node**
- Deleting a node and all its subnodes: **deleteall /node**

Quiz

1. (Single-choice) HDFS is developed based on the requirements of accessing and processing ultra-large files in streaming data mode. It features high fault tolerance, reliability, scalability, and throughput. Which of the following read/write task types is suitable for HDFS? ()
 - A. Write once, read few times
 - B. Write multiple times, read few times
 - C. Write multiple times, read multiple times
 - D. Write once, read multiple times
2. Why is it recommended that ZooKeeper be deployed in an odd number?

- Answer:
 - 1. D
 - 2. The DR capability is the same, but the deployment cost is low.

Quiz

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

- Answer:
 - 3. Blocks are larger than disks to minimize addressing overhead. If the block is large enough, the time taken to transfer data from the disk is much longer than the time required to locate the start position of the block. However, the value cannot be too large because a map usually processes data in only one block. If the number of maps is too small, the job running speed is slow.
 - 4. When data is being written, the written data is immediately visible in the namespace. When more than one block is written or the write ends, the data is visible to a new reader. The block being written cannot be seen by other readers.

Summary

- The distributed file system is an effective solution for large-scale data storage in the era of big data. The open source HDFS implements GFS and distributed storage of massive data by using a computer cluster formed by inexpensive hardware.
- HDFS is compatible with inexpensive hardware devices, stream data read and write, large data sets, simple file models, and powerful cross-platform compatibility. However, HDFS has its own limitations. For example, it is not suitable for low-latency data access, cannot efficiently store a large number of small files, and does not support multi-user write and arbitrary file modification.
- "Block" is the core concept of HDFS. A large file is split into multiple blocks. HDFS adopts the abstract block concept, supports large-scale file storage, simplifies system design, and is suitable for data backup.
- The ZooKeeper distributed service framework is used to solve some data management problems that are frequently encountered in distributed applications and provide distributed and highly available coordination service capabilities.

Acronyms and Abbreviations

- POSIX: Portable Operating System Interface
- NN: NameNode
- DN: DataNode
- NS: Namespace
- RPC: remote procedure call
- ZKFC: ZooKeeper Failover Controller
- HA: High Availability
- JN: JournalNode, which stores the Editlog generated by the active NameNode.

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



HBase — Distributed Database &
Hive — Distributed Data Warehouse



Foreword

- This chapter describes the open source non-relational distributed database HBase and distributed data warehouse Hive. Apache HBase can meet the requirements of large-scale real-time data processing applications. Apache Hive helps read, write, and manage large data sets residing in distributed storage using SQL. Structures can be projected onto stored data. The command line tool and JDBC driver are provided to connect users to Hive.

Objectives

- Upon completion of this course, you will be able to:
 - Master HBase-related concepts and data models.
 - Master the HBase architecture.
 - Be familiar with HBase performance optimization.
 - Be familiar with the basic shell operations of HBase.
 - Master Hive use cases and basic principles.
 - Master Hive architecture and its running process.
 - Be familiar with common Hive SQL statements.

- This slide introduces the following parts: 1. HBase definition as well as its use cases and positioning 2. Data storage knowledge about KeyValue (This part is applicable to other components) 3. RegionServer, HMaster, and ZooKeeper services (data process of HBase) 4. Data writing and reading process of HBase 5. Enhancements of FusionInsight HBase

Contents

1. HBase — Distributed Database

- HBase Overview and Data Models
- HBase Architecture
- HBase Performance Tuning
- Common Shell Commands of HBase

2. Hive — Distributed Data Warehouse

HBase Overview

- HBase is a column-based distributed storage system that features high reliability, performance, and scalability.
 - HBase is suitable for storing BigTable data (which can contain billions of rows and millions of columns) and allows real-time data access.
 - HDFS, as the file storage system, provides a distributed database system that supports real-time read and write operations.
 - HBase uses ZooKeeper for its collaboration service.

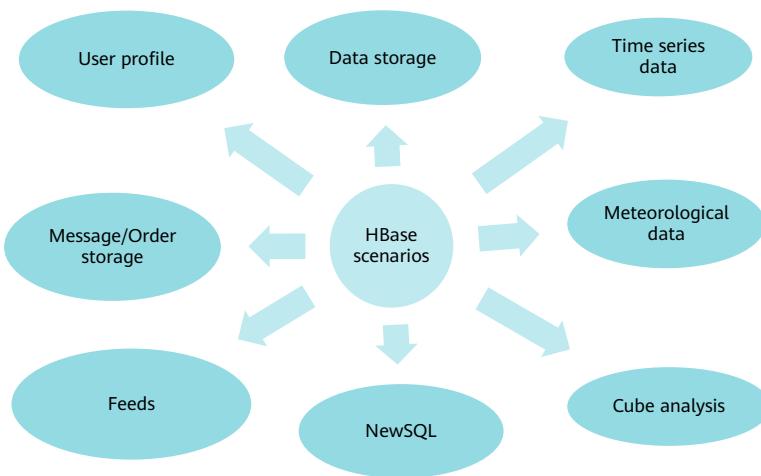
- BigTable: bigtable structure
 - A BigTable is a sparse, distributed, persistent, multidimensional sorted map with row keys, column keys, and timestamp indexes. Each value is a continuous byte array.
- ZooKeeper has the following functions:
 - 1. Distributed lock
 - 2. Event monitoring
 - 3. Storage of RegionServer data of HBase, acting as a micro database

HBase vs. Relational Databases

- 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: the row key. All access methods in HBase can be accessed using the row key or row key scanning, which ensures that systems run properly.
 - Data maintenance: During updates in 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.

- HBase: compatible with structured and unstructured data, large capacity, high concurrency, low latency, and cost-effectiveness
- 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 row key or row key scanning, which ensures that systems run properly.
 - Data maintenance: In relational databases, the most recent value replaces the original value in the record during updates. 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.

Use Cases of HBase



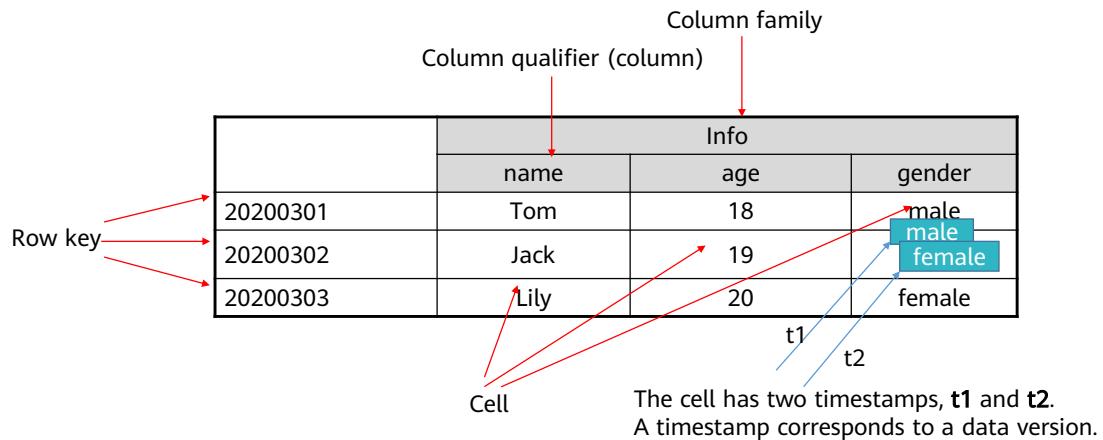
- Object storage: 1 B to 100 MB object storage (images, web pages, texts, and news) - mass storage
- Time series data: time series data (sensors, monitoring data, and Candlestick chart data) - high concurrency/mass storage
- Meteorological data: satellite orbit and meteorological data - high concurrency/mass storage
- Cube analysis: real-time reports - high concurrency/mass storage
- NewSQL: metabase and index query - SQL, secondary index, and dynamic column
- Feeds: moments - high concurrent requests
- Message/order storage: storage of chat messages and orders - forcible synchronization of massive amounts of data
- User profile: user characteristic storage - sparse matrix of tens of thousand columns
- Compatible with structured and unstructured data, large data capacity, high concurrency, low latency, and cost-effectiveness

HBase Data Model

- Applications store data in HBase as tables. A table consists of 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	More
	column1	column2	column1	column2	column3	column1	
Key1	t1:abc T2:gdfx			t4:hello t3:world			
Key2		t2:xxzz t1:yyxx					

HBase Table Structure



- In HBase, a cell needs to be determined based on the row key, column family, column qualifier, and timestamp. Therefore, a cell can be regarded as a four-dimensional coordinate, [row key, column family, column qualifier, timestamp].

HBase Table Structure

- Table: HBase uses tables to organize data. A table consists of rows and columns, and a column is divided into several column families.
- Row: Each HBase table consists of multiple rows, with each row identified by a row key.
- Column family: An HBase table is divided into multiple column families, which are basic access control units.
- Column qualifier: Data in a column family is located using column qualifiers (or columns).
- Cell: In an HBase table, a cell is determined by its row, column family, and column qualifier. The data stored in a cell has no type and is viewed as a byte array — `byte[]`.
- Timestamp: Each cell stores multiple versions of the same data. These versions are indexed using timestamps.

Conceptual View of Data Storage

- There is a table named **wehtable** that contains two column families: **contents** and **anchor**. In this example, **anchor** has two columns (**anchor:aa.com** and **anchor:bb.com**), and **contents** has only one column (**contents:html**).

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:aa.com= "CNN"
"com.cnn.www"	t8		anchor:bb.com= "CNN.com"
"com.cnn.www"	t6	contents:html=<html>..."	
"com.cnn.www"	t5	contents:html=<html>..."	
"com.cnn.www"	t3	contents:html=<html>..."	

Physical View of Data Storage

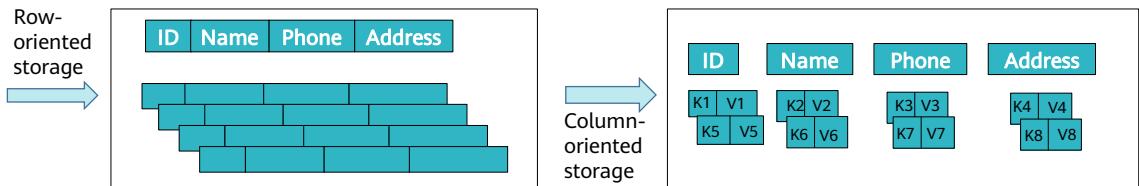
- In the conceptual view, a table can be considered as a collection of sparse rows. Physically, however, it is differentiated by column family storage. New columns can be added to a column family without being declared.

Row Key	Time Stamp	ColumnFamily anchor
"com.cnn.www"	t9	anchor:aa.com= "CNN"
"com.cnn.www"	t8	anchor:bb.com= "CNN.com"

Row Key	Time Stamp	ColumnFamily contents
"com.cnn.www"	t6	contents:html="<html>..."
"com.cnn.www"	t5	contents:html="<html>..."
"com.cnn.www"	t3	contents:html="<html>..."

Row- and Column-oriented Storage

- Row-oriented storage refers to data stored by rows in an underlying file system. Generally, a fixed amount of space is allocated to each row.
 - Advantages: Data can be added, modified, or read by row.
 - Disadvantage: Some unnecessary data is obtained when a column is queried.
- Column-oriented storage refers to data stored by columns in an underlying file system.
 - Advantage: Data can be read or calculated by column.
 - Disadvantage: When a row is read, multiple I/O operations may be required.



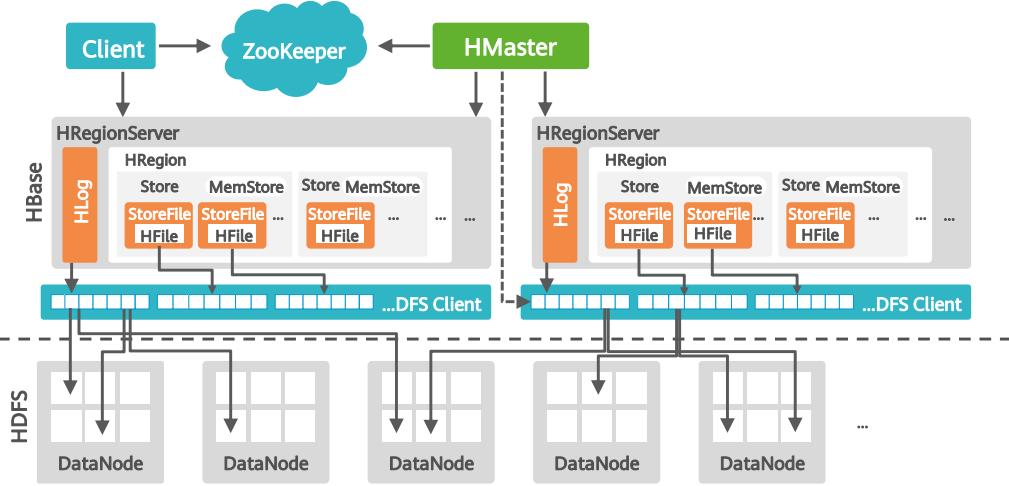
Contents

1. HBase — Distributed Database

- HBase Overview and Data Models
- HBase Architecture
 - HBase Performance Tuning
 - Common Shell Commands of HBase

2. Hive — Distributed Data Warehouse

HBase Architecture



- ZooKeeper provides distributed coordination services for each process in HBase clusters. Each RegionServer is registered with ZooKeeper so that active Master can obtain the health status of each RegionServer.
- Client communicates with Master and RegionServer using the remote procedure call (RPC) mechanism of HBase. Client communicates with Master for management purposes and communicates with RegionServer for data operation purposes.
- RegionServer provides the read and write services of table data. It functions as a data processing and computing unit in HBase. Typically, RegionServer is deployed together with DataNodes of the HDFS cluster to implement data storage.
- In HA mode, HMaster consists of an active master and a standby master.
 - Active Master manages RegionServer in HBase, including creating, deleting, modifying, and querying a table, balances the load of RegionServer, adjusts the distribution of a region, splits the region and allocates the split region, and migrates the region after RegionServer expires.
 - Standby master takes over services when the active master is faulty. The original active master demotes to the standby master after the fault is rectified.
- HDFS provides highly reliable file storage services for HBase. All HBase data is stored in HDFS.

Client

- The client provides an interface for accessing HBase, but does not directly read data from the active HMaster server. Instead, the client obtains the region location information from ZooKeeper and reads the data from HRegionServer directly. Most clients never communicate with HMaster, which makes the HMaster load light.
- The client maintains the location information of the accessed regions in the cache to accelerate subsequent data access.

- The client queries the **hbase:meta** table and then determines the location of the region. After locating the desired region, the client accesses the region and initiates a read/write request directly.
- The HBase architecture consists of the following functional components:
 - Library functions (linking to each client)
 - One HMaster server
 - Multiple HRegionServers
- MemStore: When the size of MemStore in RegionServer reaches the upper limit, RegionServer flushes data in MemStore to HDFS.
- StoreFile: As more data is inserted, multiple StoreFiles are generated in a Store. When the number of StoreFiles reaches the upper limit, RegionServer merges multiple StoreFiles into one big StoreFile.

HMaster

- The HMaster server manages tables and regions by performing the following operations:
 - Manages users' operations on tables, such as adding, deleting, modifying, and querying.
 - Implements load balancing between different RegionServers.
 - Adjusts the distribution of regions after they are split or merged.
 - Migrates the regions on faulty RegionServers.
- HMaster HA
 - ZooKeeper can help elect an HMaster as the primary management node of the cluster and ensure that there is only one HMaster running at any time, preventing single points of failure (SPOF).

- HMaster manages and maintains the partition information in HBase tables, maintains HRegionServers, allocates regions, and balances loads.

HRegionServer

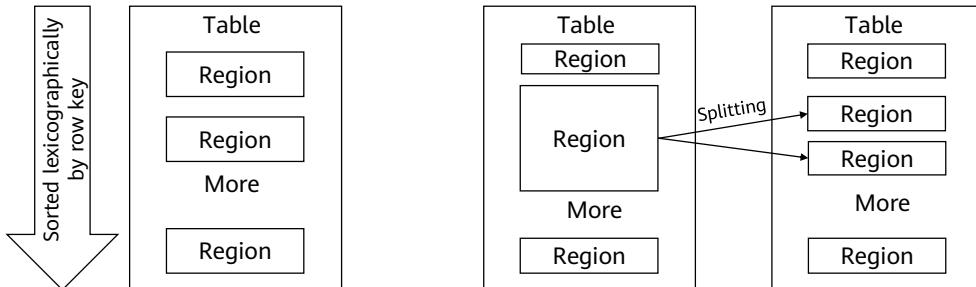
- HRegionServer is the core module of HBase.
 - It maintains the regions allocated.
 - It processes read and write requests from clients.

HBase Table

- Table (HBase table)
 - Region (Regions for the table)
 - Store (Store per ColumnFamily for each Region for the table)
 - MemStore (MemStore for each Store for each Region for the table)
 - StoreFile (StoreFiles for each Store for each Region for the table)
 - Block (Blocks within a StoreFile within a Store for each Region for the table)

Region

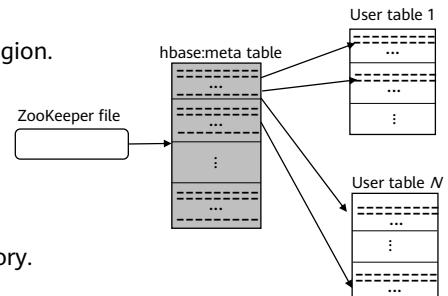
- An HBase table has only one region at the beginning. When the size of the region exceeds the preconfigured threshold, HBase splits the region.
- The region splitting operation is almost instantaneous because after the splitting, the region still reads the original storage file. The region reads the new file only after the storage file is asynchronously written to an independent file after the splitting process is complete.



- Region splitting policies:
 - ConstantSizeRegionSplitPolicy
 - IncreasingToUpperBoundRegionSplitPolicy
 - SteppingSplitPolicy
 - KeyPrefixRegionSplitPolicy
 - DelimitedKeyPrefixRegionSplitPolicy
 - DisabledRegionSplitPolicy
- Region combination
 - Region splitting is mainly used to improve performance, but region combination is mainly used to maintain the system.

Region Positioning

- Regions are classified into meta regions and user regions.
- Meta regions record the routing information of each user region.
- To read and write region routing information, perform the following steps:
 - Find the addresses of meta regions.
 - Find the addresses of user regions based on meta regions.
- To accelerate access, the **hbase:meta** table is saved in memory.
- Assume that each row (a mapping entry) in the **hbase:meta** table occupies approximately 1 KB of memory, and the maximum size of each region is 128 MB.
- In the two-layer structure, 2^{17} (128 MB/1 KB) regions can be saved.



- Metadata table, also called **hbase:meta** table, stores the mapping between regions and region servers.
- In versions later than HBase 0.96, HBase removes the ROOT** table and retains only **hbase:meta**.

Data Read and Write Process

- When you write data, it is allocated to the corresponding HRegionServer for execution.
- Your data is first written to HLog, then to MemStore, and finally to disks to generate a StoreFile.
- The **commit()** invocation returns the data to the client only after the operation is written to HLog.
- When you read data, the HRegionServer first accesses the MemStore cache. If the MemStore cache cannot be found, the HRegionServer searches StoreFile on the disk.

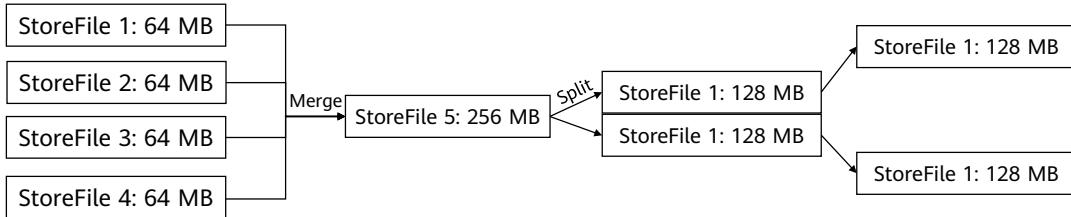
Cache Refreshing

- The system periodically writes the content in the MemStore cache to the StoreFile file on the disk, clears the cache, and writes a tag in the HLog.
- A new StoreFile file is generated each time data is written. Therefore, each Store contains multiple StoreFile files.
- 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 refresh operation. If an update is detected, the HLog file is written to MemStore and then StoreFile to provide services for users.

- Scenarios that trigger MemStore Flush:
 - When the size of a MemStore reaches the value (128 MB by default) specified for **HBase.hregion.memstore.flush.size**, the flush operation is triggered for all MemStores.
 - When the memory usage of a MemStore reaches the value of **HBase.regionserver.global.memstore.upp -erLimit**, the flush operation is triggered. The flush sequence is based on MemStore memory usage in descending order until the MemStore memory usage is less than the value of **HBase.regionserver.global.memstore.lowerLimit**.
 - When there are more WAL files than the value of **HBase.regionserver.max.logs**, MemStore is flushed to disks to reduce the number of WAL files. The oldest MemStore is flushed first until the number of logs is less than the value of **HBase.regionserver.max.logs**.
 - HBase periodically flushes MemStore, with a default interval of 1 hour. This prevents MemStore data from persisting over a long period of time.
 - Manual execution of the flush operation

Store Implementation

- Store is the core of HRegionServer.
- StoreFile merge: A new StoreFile is generated each time data is flushed. Due to the large number of StoreFiles, search speeds are affected. The **Store.compact()** function is called to merge multiple StoreFiles into one.
- When a single StoreFile is too large, the splitting operation is triggered. One parent region is split into two sub-regions.



- The merge operation is performed only when the number of StoreFiles reaches a threshold, because it is consuming a large number of resources.

HLog Implementation

- In a distributed environment, you need to consider system errors. HBase uses HLog to ensure system recovery.
- The HBase system configures an HLog file, a write-ahead log (WAL), for each HRegionServer.
- 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 flushed to the disk only after the logs corresponding to the data cached in the MemStore are written to the disk.

HLog Implementation

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

Contents

1. HBase — Distributed Database

- HBase Overview and Data Models
- HBase Architecture
- **HBase Performance Tuning**
- Common Shell Commands of HBase

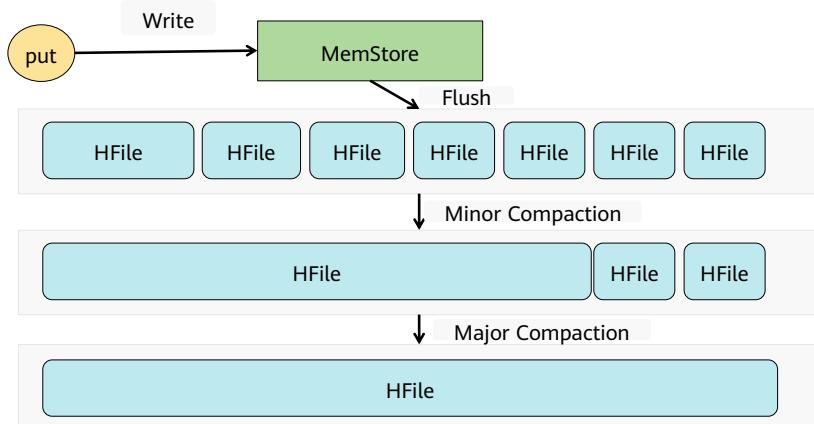
2. Hive — Distributed Data Warehouse

Compaction

- Read latency lengthens as the number of HFiles increases.
- Compaction reduces the number of small files (HFiles) in the same column family of the same region to improve read performance.
- Compaction is classified into minor compaction and major compaction.
 - Minor compaction: small-scale compaction. There are limits on the minimum and maximum number of files. Typically, small files in a continuous time range are merged.
 - Major compaction: all HFile files under the ColumnFamily of the region.
 - Minor compaction complies with a certain algorithm when selecting files.

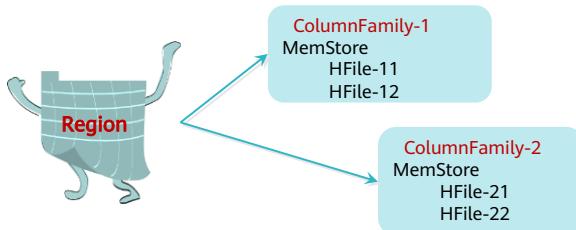
- As time passes, the number of HFiles increases because service data flows to the HBase cluster. The same query opens more files, so the query latency increases.
- Compaction checks all HFiles in the store one by one and excludes the files that do not meet the requirements.
 - Files that are being compacted and that are updated after the compacted files (with a larger sequence ID) are excluded.
 - If the size of a single file is greater than the value of **hbase.hzstore.compaction.max.size**(default maximum **Long** value), the file is excluded. Otherwise, a large number of I/O resources are consumed.
- The files that are excluded are called candidate files. HBase determines whether major compaction conditions are met. If the conditions are met, HBase merges all files. If any of the following conditions is met, major compaction is executed:
 - The user forcibly executes major compaction.
 - Candidate files have not been compacted for a long time and the number of candidate files is less than the value of **hbase.hstore.compaction.max** (10 by default).
 - The store contains reference files. Reference files are temporary files generated during region splitting. They are simple reference files and must be deleted during compaction.
- If the major compaction condition is not met, minor compaction is executed.

Compaction



OpenScanner

- In the OpenScanner process, two different scanners are created to read HFile and MemStore data.
 - The scanner corresponding to HFile is StoreFileScanner.
 - The scanner corresponding to MemStore is MemStoreScanner.



- Before locating the corresponding RegionServer and Region of RowKeys, open a Scanner to search for data. Because a Region contains MemStores and HFiles, open a Scanner for each of them to read data in MemStores and HFiles separately.

BloomFilter

- BloomFilter optimizes random read scenarios, that is, the **Get** scenario. It can be used to quickly determine whether a user data record exists in a large dataset (most data in the dataset cannot be loaded to the memory).
- BloomFilter can misjudge whether a data record exists. However, the judgment result of "The subscriber data XXXX does not exist." is reliable.
- HBase BloomFilter data is stored in HFiles.

- If BloomFilter determines that the data does not exist, then it definitely does not exist. But if it determines that the data does exist, then it's important to remember that the data may not actually exist.

Row Key

- Row keys are stored in alphabetical order. Therefore, when designing row keys, you need to fully utilize the sorting feature to store frequently read data together and store recently accessed data together.
- For example, if 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 achieves hits when being read.
- HBase has only one index for row keys.
- You can access rows in an HBase table in any of the following ways:
 - Single row key
 - Row key interval
 - Full table scan

Creating HBase Secondary Index

- Hindex secondary index
 - Hindex is an HBase secondary index compiled in Java by Huawei. It is compatible with Apache HBase 0.94.8. The features are as follows:
 - Multiple table indexes
 - Multiple column indexes
 - Index based on some column values

Contents

1. HBase — Distributed Database

- HBase Overview and Data Models
- HBase Architecture
- HBase Performance Tuning
- **Common Shell Commands of HBase**

2. Hive — Distributed Data Warehouse

Common Shell Commands of HBase

- **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.
- **enable/disable**: enables or disables a table.
- **drop**: deletes a table.

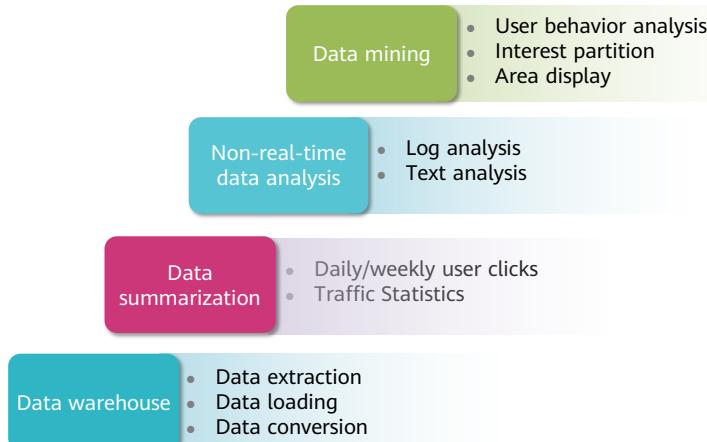
Contents

1. HBase — Distributed Database
2. **Hive — Distributed Data Warehouse**
 - Hive Overview
 - Hive Functions and Architecture
 - Basic Hive Operations

Hive Overview

- Hive is a data warehouse tool that runs on Hadoop and supports PB-level distributed data query and management.
- Hive features:
 - Flexible extract, transform, load (ETL)
 - Support for multiple compute engines, such as Tez and Spark
 - Direct access to HDFS files and HBase
 - Easy to use and program

Use Cases of Hive



- Hive is built on Hadoop for static batch processing. Hadoop has long latency and consumes a large number of resources during job submission and scheduling.
- Hive does not support quick query of data in a large-scale data set. For example, Hive has minutes of latency when querying data on a dataset of hundreds of MB. Therefore, Hive is inapplicable for scenarios that require low latency, for example, online transaction processing (OLTP).
- Hive is not designed for OLTP. Therefore, it does not support real-time query and row-based data update. Hive is most applicable to batch processing jobs of large data sets, for example, network log analyses.

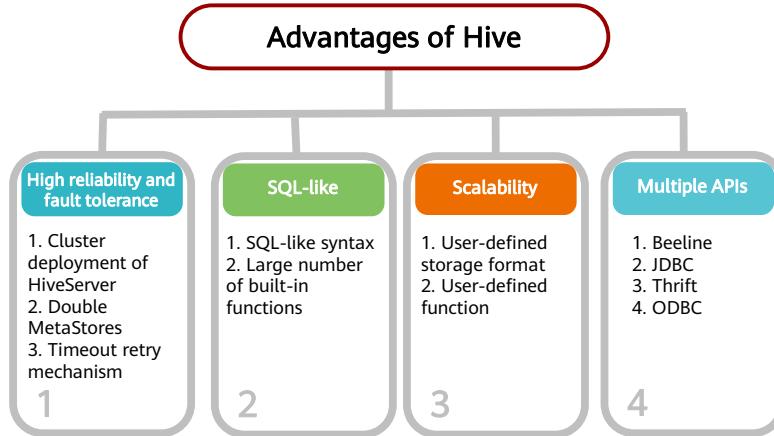
Hive vs. Conventional Data Warehouses

	Hive	Conventional Data Warehouse
Storage	HDFS is used to store data. Theoretically, infinite expansion is possible.	Clusters are used to store data, which has 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.
Execution engine	The default execution engine is Tez.	More efficient algorithms can be selected to perform queries and 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 conventional data warehouse.	When the data volume is small, the data processing speed is high. When the data volume is large, the speed decreases sharply.

Hive vs. Conventional Data Warehouses

	Hive	Conventional Data Warehouse
Index	Low efficiency	High efficiency
Usability	Application models need to be self-developed. The flexibility is high, but the usability is low.	Integrates a set of mature report solutions 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.
Environment dependency	Low dependency on hardware, applicable to common machines	Highly dependent on high-performance business servers
Price	Open source	Expensive

Advantages of Hive

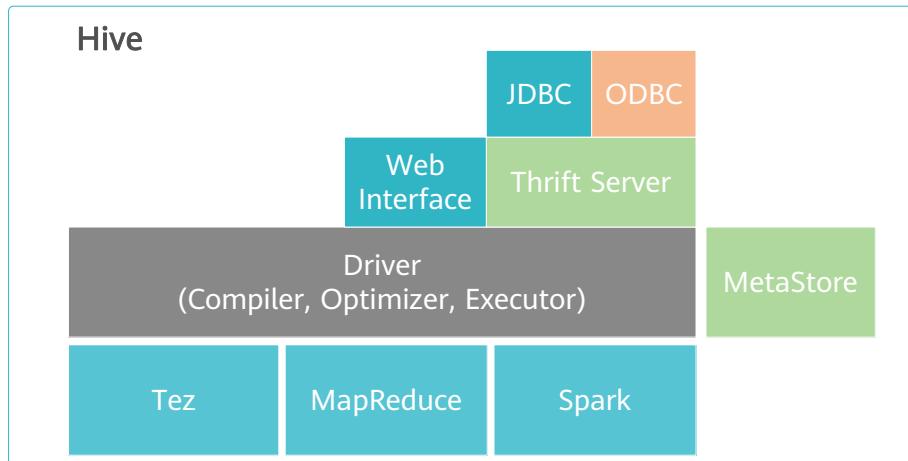


- Main process for HiveServer Hive to provide SQL services for external systems.
- MetaStore Hive is a process that provides metadata information and can be invoked by components such as HiveServer, SparkSQL, and Oozie.
- Beeline: Hive CLI client
- JDBC: Java unified database interface
- Thrift is a serialization and communication protocol.
- Python is a dynamic language.
- ODBC is a standard database interface based on C/C++.

Contents

1. HBase — Distributed Database
2. **Hive — Distributed Data Warehouse**
 - Hive Overview
 - Hive Functions and Architecture
 - Basic Hive Operations

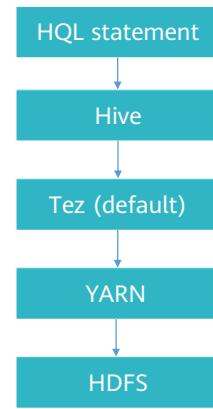
Hive Architecture



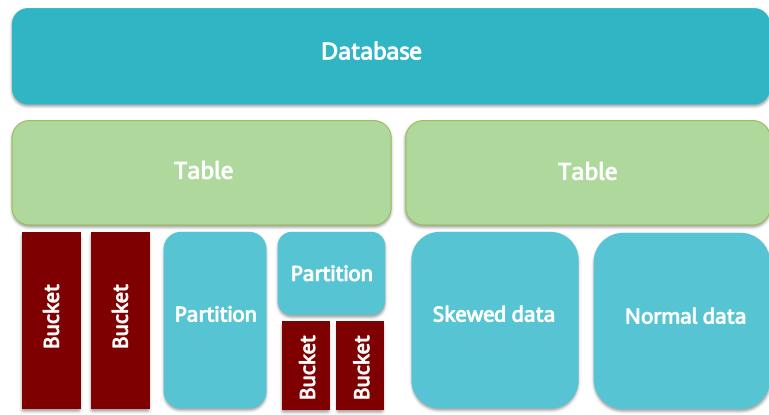
- MetaStore: stores metadata such as tables, columns, and partitions.
- Driver: manages the lifecycle of HQL execution and participates in the entire Hive job execution.
- Compiler: translates HQL statements into a series of interdependent Map or Reduce jobs.
- Optimizer: is classified into logical optimizer and physical optimizer to optimize HQL execution plans and MapReduce jobs, respectively.
- Executor: runs Map or Reduce jobs based on job dependencies.
- ThriftServer: functions as the servers of JDBC and ODBC, provides Thrift APIs, and integrates Hive and other applications.
- Clients: include command line interface Beeline, and JDBC/ODBC interfaces, and provide interfaces for user access.
- Hive statements are ultimately executed on Tez, MapReduce, or Spark.

Hive Running Process

- A client submits HQL commands.
- Tez executes queries.
- YARN allocates resources to applications in the cluster and authorizes Hive jobs in the YARN queue.
- Hive updates data in HDFS or the Hive data warehouse based on table type.
- Hive returns the query result through the JDBC connection.



Data Storage Model of Hive



- Database: If the database is not specified when a table is created, the default database is used.
- Table: corresponds to a directory on HDFS.
- Partition: corresponds to a subdirectory in the directory where the table is located.
- Bucket: corresponds to a file under the path where a table or a partition resides.
- Skewed data: scenarios where data is converged in some field values, for example, when data is classified by city, 80% of data comes from a big city.
- Normal data: Skewed data does not exist.

Hive Partition and Bucket

- Partition: Data tables can be partitioned based on the value of a certain field.
 - Each partition is a directory.
 - The number of partitions is not fixed.
 - Partitions or buckets can be created within partitions.
- Bucket: Data can be stored in different buckets.
 - Each bucket is a file.
 - The number of buckets is specified when creating a table. The buckets can be sorted.
 - Data is hashed based on the value of a field and then stored in a bucket.

- Hive can organize tables or partitions to buckets, that is, buckets are divided by smaller data ranges.
- A column can be bucketed using Hive. Hive hashes the column value and divides the value by the number of buckets to determine the bucket where the record is stored. Buckets make query and sampling more efficient.
- Bucketing applies to the following scenarios: (1) Data sampling; (2) Improving the efficiency of some query operations, such as mapside join.
- When a table contains a large amount of data, you can partition the table to facilitate local data query. For example, you can partition the table by time or region and store data with the same property to the same disk block, improving the query efficiency.
- Partition table: CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);

Hive Data Storage Model: Managed and External Tables

- Hive can create managed and external tables.
 - After a Hive table is created, Hive moves data to the data warehouse directory.
 - If all processing is performed by Hive, you are advised to use managed tables.
 - If you want to use Hive and other tools to process the same data set, you are advised to use external tables.

	Managed Table	External Table
DROP	Metadata and data are deleted together.	Only metadata is deleted.

- When creating an internal table, Hive moves data to the path to which the data warehouse points. If an external table is created, only the data path is recorded and the data location is not changed.
- When a table is deleted, the metadata and data of an internal table are deleted together. However, for an external table, only the metadata is deleted. In this way, external tables are more secure, data organization is more flexible, and source data can be shared easily.

Functions Supported by Hive

- Built-in Hive functions
 - Mathematical functions, such as **round()**, **floor()**, **abs()**, and **rand()**.
 - Date functions, such as **to_date()**, **month()**, and **day()**.
 - String functions, such as **trim()**, **length()**, and **substr()**.
- User-defined functions (UDFs)

- **round**: round off; **floor**: round down; **abs**: absolute value; **rand**: random number
- **to_date**: returns date. **month**: returns the month in the date; **day**: returns date.
- `hive> select day('2011-12-08 10:03:01') from dual;8`

- If built-in functions do not meet users' requirements, Hive supports user-defined functions.
- UDFs are used to meet the requirement of one-to-one mapping.

Contents

1. HBase — Distributed Database
2. **Hive — Distributed Data Warehouse**
 - Hive Overview
 - Hive Functions and Architecture
 - Basic Hive Operations

Usage of Hive

- Running HiveServer2 and Beeline

```
$ $HIVE_HOME/bin/hiveserver2  
$ $HIVE_HOME/bin/beeline -u jdbc:hive2://$HS2_HOST:$HS2_PORT
```

- Running Hcatalog

```
$ $HIVE_HOME/hcatalog/sbin/hcat_server.sh
```

- Running WebHCat (Templeton):

```
$ $HIVE_HOME/hcatalog/sbin/webhcat_server.sh
```

- DDL is short for data definition language. DDL operations are performed on metadata. The following operations are involved:

- Create/Drop/Alter Database;
- Create/Drop/Truncate Table;
- Alter Table/Partition/Column;
- Create/Drop/Alter View;
- Create/Drop Index;
- Create/Drop Function;
- Show;
- Describe

Hive SQL Overview

- DDL-Data Definition Language:
 - Creates tables, modifies tables, deletes tables, partitions, and data types.
- DML-Data Management Language:
 - Imports and exports data.
- DQL-Data Query Language:
 - Performs simple queries.
 - Performs complex queries such as **Group by**, **Order by**, and **Join**.

- DDL is short for data definition language. DDL operations are performed on metadata. The following operations are involved:
 - Create/Drop/Alter Database;
 - Create/Drop/Truncate Table;
 - Alter Table/Partition/Column;
 - Create/Drop/Alter View;
 - Create/Drop Index;
 - Create/Drop Function;
 - Show;
 - Describe

DDL Operations

-- Create a table:

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
```

```
hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

-- Browse tables:

```
hive> SHOW TABLES;
```

-- Describe a table:

```
hive> DESCRIBE invites;
```

-- Modify a table:

```
hive> ALTER TABLE events RENAME TO 3koobecaf;
hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
```

- `hive> ALTER TABLE events RENAME TO 3koobecaf;`
- `hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);`

DML Operations

-- Load data to a table:

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
```

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION  
(ds='2008-08-15');
```

-- Export data to HDFS:

```
EXPORT TABLE invites TO '/department';
```

- hive> ALTER TABLE events RENAME TO 3koobecaf;
- hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
- Exporting data to HDFS

DQL Operations

--SELECTS and FILTERS

```
hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';
```

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='2008-08-15';
```

--GROUP BY

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(*) WHERE a.foo > 0 GROUP BY a.bar;
hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(*) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;
```

- hive> ALTER TABLE events RENAME TO 3koobecaf;
- hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
- Exporting data to HDFS

DQL Operations

--MULTITABLE INSERT

```
FROM src
INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100
INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key < 200;
```

--JOIN

```
hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar,
t1.foo, t2.foo;
```

--STREAMING

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSFORM(a.foo, a.bar) AS (oof, rab) USING
'/bin/cat' WHERE a.ds > '2008-08-09';
```

- hive> ALTER TABLE events RENAME TO 3koobecaf;
- hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
- Exporting data to HDFS

Quiz

1. (Single-choice) In which form is data stored in HBase? ()
 - A. Integer
 - B. Long
 - C. String
 - D. Byte[]

- Answer: D. The data stored in a cell does not have a data type and is always treated as a byte array: byte[].

Quiz

2. (Single-choice) What is the basic unit of HBase distributed storage? ()
 - A. Region
 - B. Column family
 - C. Column
 - D. Cell

- Answer: A

Quiz

3. (Multiple-choice) Which of the following application scenarios can Hive be used?
()
- A. Online data analysis in real time
 - B. Data mining (including user behavior analysis, region of interest, and area display)
 - C. Data summary (daily/weekly user clicks and click ranking)
 - D. Non-real-time analysis (log analysis and statistical analysis)

- Answer: BCD

Quiz

4. (Single-choice) Which of the following statements is true about basic SQL operations of Hive? ()
- A. Use keywords "external" and "internal" when you create external tables and normal tables, respectively.
 - B. The location information must be specified when an external table is created.
 - C. When data is loaded to Hive, the source data must be a path in HDFS.
 - D. Column separators can be specified when a table is created.

- Answer: D

Summary

- This chapter describes the HBase distributed database and Hive distributed data warehouse in detail.
- HBase is an open source implementation of BigTable. Like BigTable, HBase supports a large amount of data and distributed concurrent data processing. It expands easily, offers support for dynamic scaling, and can be used on inexpensive devices. HBase is a mapping table that stores data in a sparse, multi-dimensional, and persistent manner. It uses row keys, column families, and timestamps for indexing, and each value is an unexplained string.
- Hive is a distributed data warehouse. This chapter describes the use cases, basic principles, architecture, running process, and common Hive SQL statements of Hive.

Acronyms and Abbreviations

- ETL: extract, transform, and load, a process that involves extracting data from sources, transforming the data, and loading the data to final targets
- JDBC: Java Database Connectivity
- ODBC: Open Database Connectivity
- UDF: user-defined functions

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



ClickHouse — Online Analytical Processing Database Management System



Foreword

- This chapter describes ClickHouse, an online analytical processing (OLAP) database management system (DBMS), including its overview, use cases, system architecture, basic features, and enhanced features.

Objectives

- Upon completion of this course, you will be able to:
 - Understand ClickHouse and its use cases.
 - Master the ClickHouse architecture and basic features.
 - Master the enhanced features of ClickHouse.

Contents

- 1. ClickHouse Overview**
2. ClickHouse Architecture and Basic Features
3. Enhanced Features of ClickHouse

ClickHouse Overview

- ClickHouse is an OLAP column-oriented database management system developed by Yandex in Russia. It was open sourced in 2016.
- ClickHouse 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 Advantages

Comprehensive DBMS functions

Column-oriented storage and data compression

Vectorized execution engine

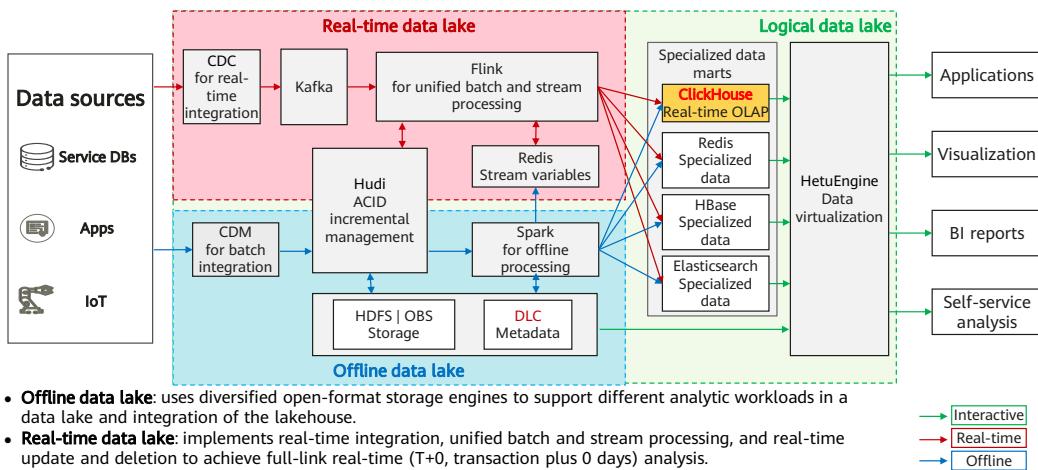
Relational model and SQL query

Data sharding and distributed query

Excellent performance

- 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.
 - Columnar databases store data in the same column together and data in different columns separately. Columnar databases are more suitable for online analytical processing (OLAP) scenarios.

Use Cases



8 Huawei Confidential



- Advantages:
 - High data compression ratio
 - Multi-core parallel computing
 - Vectorized computing engine
 - Support for nested data structure
 - Support for sparse indexes
 - Support for INSERT and UPDATE
- ClickHouse use cases:
 - Real-time data warehouse
 - The streaming computing engine (such as Flink) is used to write real-time data to ClickHouse. With the excellent query performance of ClickHouse, multi-dimensional and multi-mode real-time query and analysis requests can be responded within subseconds.
 - Offline query
 - Large-scale service data is imported to ClickHouse and constructs a large wide table with hundreds of millions to tens of billions of records and hundreds of dimensions. ClickHouse supports personalized statistics collection and continuous exploratory query and analysis at any time to assist business decision-making and provides excellent query experience.

Applicable and Inapplicable Scenarios of ClickHouse

Applicable scenarios

- Network/App traffic analysis
- User behavior analysis
- Customer estimation and profiling
- Business intelligence (BI)
- Monitoring system
- Query on wide tables and aggregation query on a single table

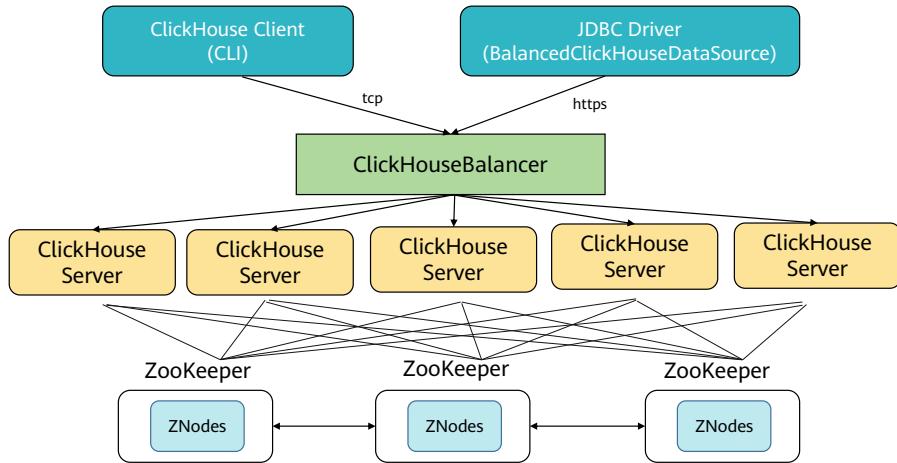
Inapplicable scenarios

- OLTP
- High-frequency key-value access
- Document storage
- Unstructured data
- Point query (sparse indexes)
- Scenarios with frequent updates or deletions

Contents

1. ClickHouse Overview
- 2. ClickHouse Architecture and Basic Features**
3. Enhanced Features of ClickHouse

ClickHouse Architecture



11 Huawei Confidential



- ClickHouseServer: provides a column-based database oriented to OLAP analysis and processing. It provides complete database management system (DBMS) functions and supports column-based storage, data compression, vectorized execution engine, relational model and SQL query, as well as data sharding and distributed query.
- ClickHouseBalancer: balances the loads on ClickHouseServers and provides high reliability capabilities. ClickHouseBalancer can balance the load of multiple ClickHouseServers to improve the reliability of application access.
- ClickHouse can be accessed by CLI and JDBC clients using either TCP or HTTP.
- ClickHouseServer provides open-source server capabilities. It implements the multi-master replica mechanism using ZooKeeper and the ReplicatedMergeTree engine.
- ClickHouse depends on ZooKeeper to implement distributed DDL execution and supports status synchronization between the active and standby nodes of the ReplicatedMergeTree table.

SQL Capability

- ClickHouse supports a declarative query language based on SQL. Supported queries include **GROUP BY**, **ORDER BY**, **FROM**, **JOIN**, **IN**, and uncorrelated subqueries. The language is basically compatible with standard syntax. It supports views and materialized views, but does not support standard **UPDATE** or **DELETE**, which is implemented through **ALTER TABLE**. Correlated (dependent) subqueries and window functions are not supported.
- Common SQL syntax of ClickHouse
 - **CREATE DATABASE**: Creating a database
 - **CREATE TABLE**: Creating a table
 - **INSERT INTO**: Inserting data into a table
 - **SELECT**: Querying table data
 - **ALTER TABLE**: Modifying a table structure
 - **DESC**: Querying a table structure
 - **DROP**: Deleting a table
 - **SHOW**: Displaying information about databases and tables

Table Engine

- Table engines play a critical role in ClickHouse to determine:
 - Where data is stored and read
 - Supported query modes
 - Whether data can be concurrently accessed
 - Whether indexes can be used
 - Whether multi-thread requests can be executed
 - Parameters used for data replication
- There are five common table engines for ClickHouse: **TinyLog**, **Memory**, **MergeTree**, **ReplacingMergeTree**, and **SummingMergeTree**.

- Distributed: supports multi-node collaborative query or write.
- ReplicatedMergeTree: supports the replication mechanism that depends on ZooKeeper.
- MergeTree: core engine

MergeTree Engine

- MergeTree and its series (*MergeTree) are the most general and powerful table engines used by ClickHouse for heavy-load tasks. They are designed to insert a large amount of data into a table. Data is quickly written as data blocks, which are asynchronously merged in the background to ensure efficient insertion and query performance.
- The following functions are supported:
 - Primary key and sparse index
 - Data partitioning
 - Replication mechanism (for ReplicatedMergeTree series)
 - Data sampling
 - Concurrent data access
 - Time To Live (TTL, for automatic deletion of data upon expiration)
 - Secondary data-skipping indexes

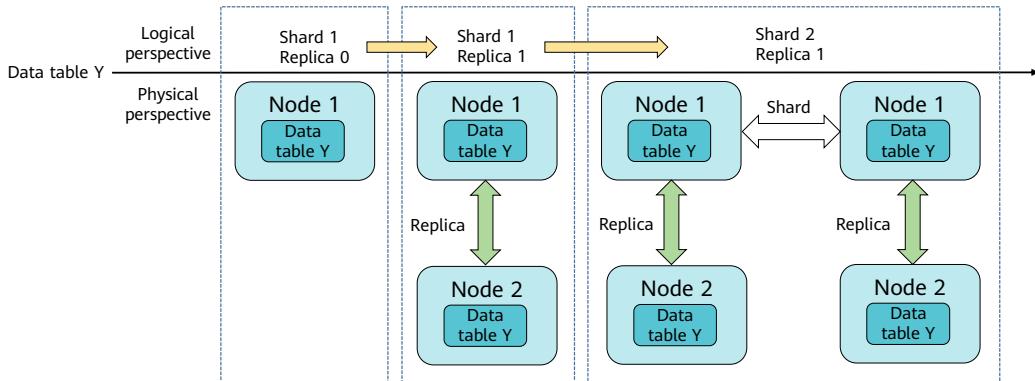
- MergeTree is the most general and powerful table engine used for heavy-load tasks. It has the following key features:
 - Data storage by partition and block based on partition keys
 - Data index is sorted based on primary keys and the **ORDER BY** sorting keys.
 - Data replication is supported by table engines prefixed with Replicated.
 - Data sampling
- When data is written, a table with this type of engine divides data into different folders based on the partitioning key. Each column of data in the folder is an independent file. A file that records serialized index sorting is created. This structure reduces the volume of data to be retrieved during data reading, greatly improving query efficiency.

Data Shards vs. Replicas

- From the data perspective
 - Assume that N ClickHouse nodes form a cluster and each node in the cluster has a table Y with the same structure. If data in table Y of N_1 is completely different from the data in table Y of N_2 , N_1 and N_2 are shards of each other. If their data is the same, they are replicas of each other. In other words, data between shards is different, but data between replicas is exactly the same.
- From the function perspective
 - The main purpose of using replicas is to prevent data loss and increase data storage redundancy, while the main purpose of using shards is to implement horizontal data partitioning.

- ClickHouse implements the replicated table mechanism based on the **ReplicatedMergeTree** engine and ZooKeeper. When creating a table, you can specify an engine to determine whether the table is highly available. Shards and replicas of each table are independent of each other.
- ClickHouse implements the distributed table mechanism based on the **Distributed** engine. Views are created on all shards (local tables) for distributed query, which is easy to use. ClickHouse has the concept of data sharding, which is one of the features of distributed storage. That is, parallel read and write are used to improve efficiency.

ClickHouse Shards and Replicas



- Cluster
 - Cluster: Cluster is a logical concept in ClickHouse. It can be defined by users as required, which is different from the general understanding of cluster. Multiple ClickHouse nodes are loosely coupled and exist independently.
- Shard
 - Shard: A shard is a horizontal division of a cluster. A cluster can consist of multiple shards.
- Replica
 - Replica: One shard can contain multiple replicas.
- Partition
 - Partition: A partition is used for local replicas and can be considered as a vertical division.
- MergeTree
 - ClickHouse has a huge table engine system. As the basic table engine of the family system, MergeTree provides functions such as data partitioning, primary indexes, and secondary indexes. When creating a table, you need to specify the table engine. Different table engines determine the final character of a data table, for example, the features of a data table, the form how data is stored, and how data is loaded.

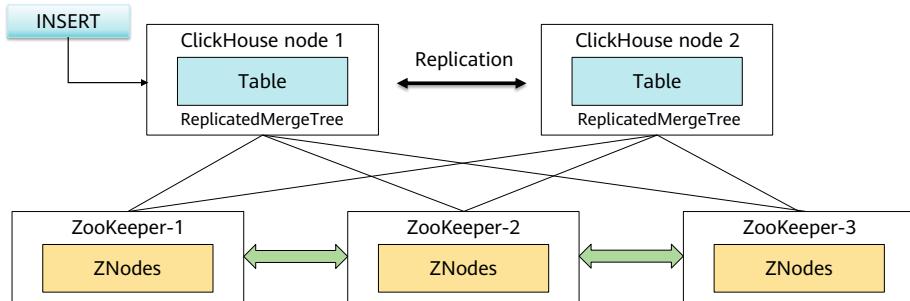
Characteristics of ClickHouse Replicas

- Table-level replicas. The replica configuration of each table can be customized based on requirements, including the number of replicas and the distributed location of replicas in the cluster.
- The distributed collaboration capability of ZooKeeper is required to implement data synchronization between multiple replicas.
- Multi-active architecture. INSERT and ALTER queries on any replica have the same effect. These operations are distributed to each replica for local execution through the coordination capability of ZooKeeper.
- Data to be written is divided into several blocks based on the value of **max_insert_block_size**. A block is the basic unit for data writing and is atomic and unique.
 - Atomicity: All data blocks are successfully written or fail to be written.
 - Uniqueness: Hash message digests are calculated based on indicators such as data sequence, row, and size to prevent data blocks from being repeatedly written due to exceptions.

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

Replica Mechanism

- The ClickHouse replica mechanism minimizes network data transmission and synchronizes data between different data centers. It can be used to build clusters with the remote multi-active multi-DC architecture.
- The replication mechanism is the basis for implementing high availability (HA), load balancing, migration, and upgrade.
- High availability: The system monitors the status of replica synchronization, identifies faulty nodes, and rectifies faults when the nodes are recovered to ensure HA of the entire service.

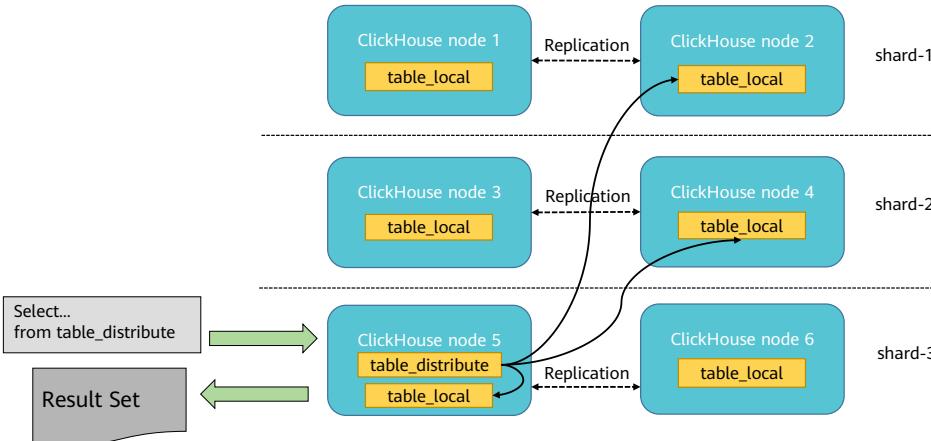


Distributed Query

- ClickHouse provides the linear expansion capability by using the sharding and distributed table mechanism.
- **Sharding:** eliminates the performance bottleneck of a single node. Data in a table is horizontally split to multiple nodes, and data on different nodes is not duplicate. In this way, users can expand ClickHouse nodes by adding shards.
- **Distributed table:** A distributed table is used to query sharded data. The distributed table engine does not store any data and is only a layer-1 proxy. It can automatically route data to each shard node in the cluster to obtain data. That is, a distributed table needs to work with other data tables.

- Data partitioning: allows a query to read as little data as possible if a partition key is specified.
- Data sharding: allows multiple machines or nodes to execute queries concurrently, implementing distributed parallel computing.

Distributed Query



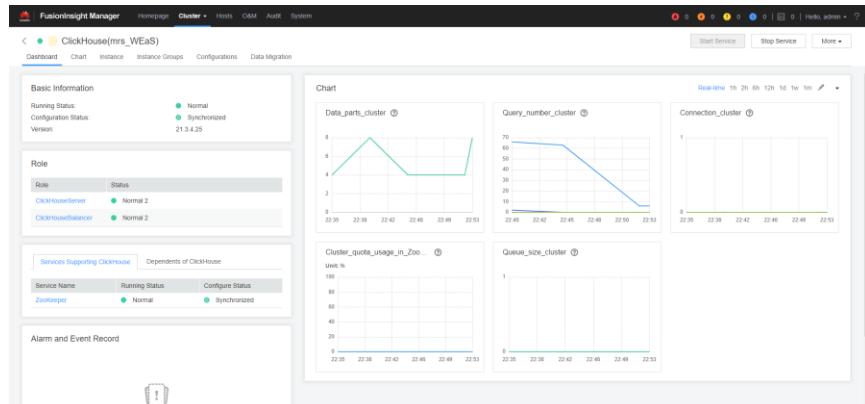
- The process of executing a query on a distributed table is as follows: After a query is sent, instances exchange their own sharded table data, aggregate the sharded table data to the same instance, and return the data to users.

Contents

1. ClickHouse Overview
2. ClickHouse Architecture and Basic Features
- 3. Enhanced Features of ClickHouse**

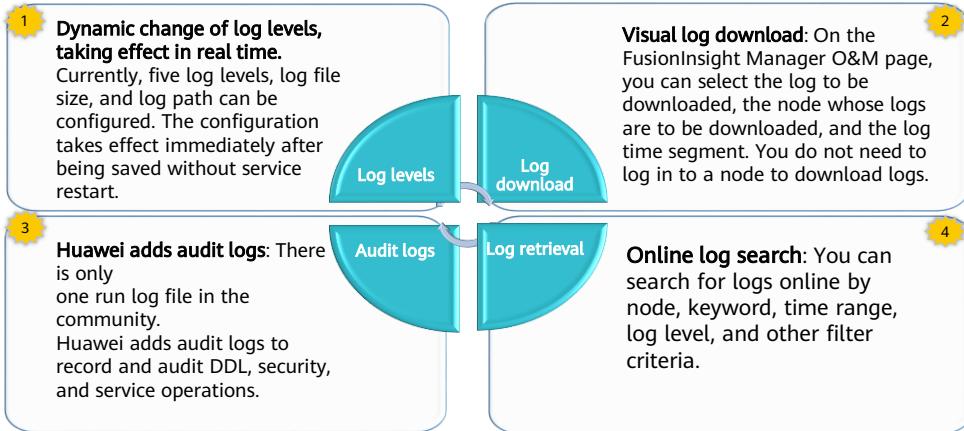
Visualized O&M

- Visualized O&M methods are provided, such as ClickHouse installation and configuration, startup and stop, client, 70+ monitoring and alarm metrics, and health check, making ClickHouse easier to maintain and operate.

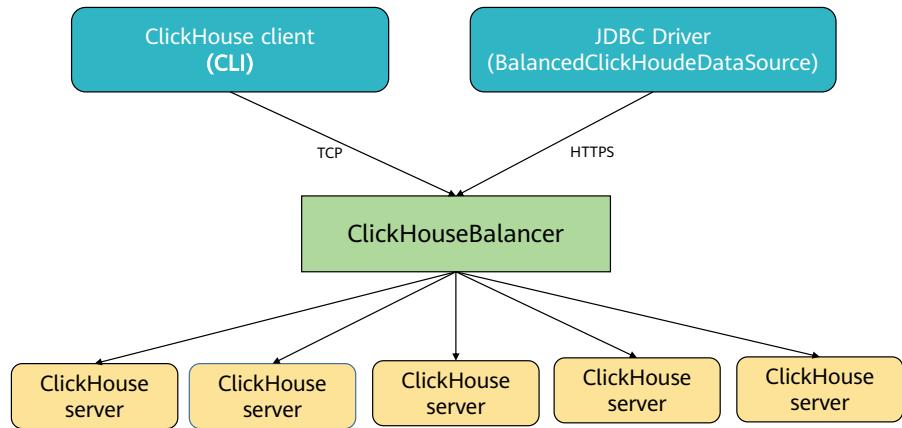


Visualized O&M

- Log levels can be configured on the GUI, and the configuration dynamically takes effect. Visual log downloading and retrieval are supported, and audit logs are added to improve fault locating efficiency.



Load Balancing



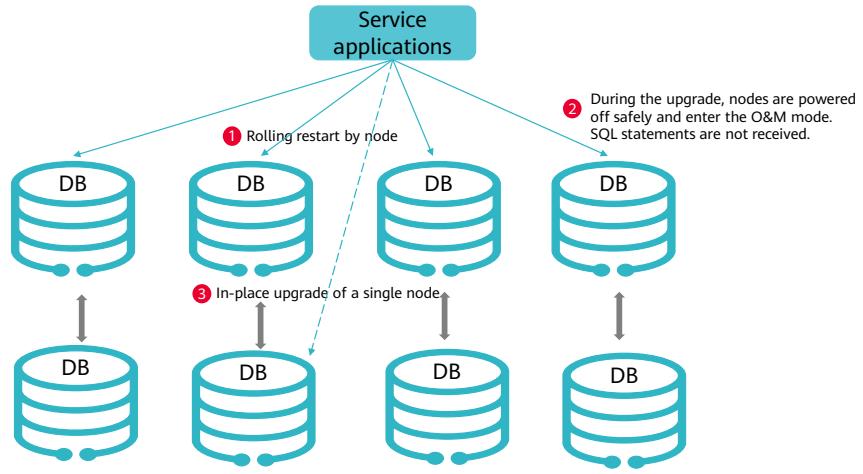
- ClickHouseBalancer: route access load balancer.

Online Scale-out and Data Migration

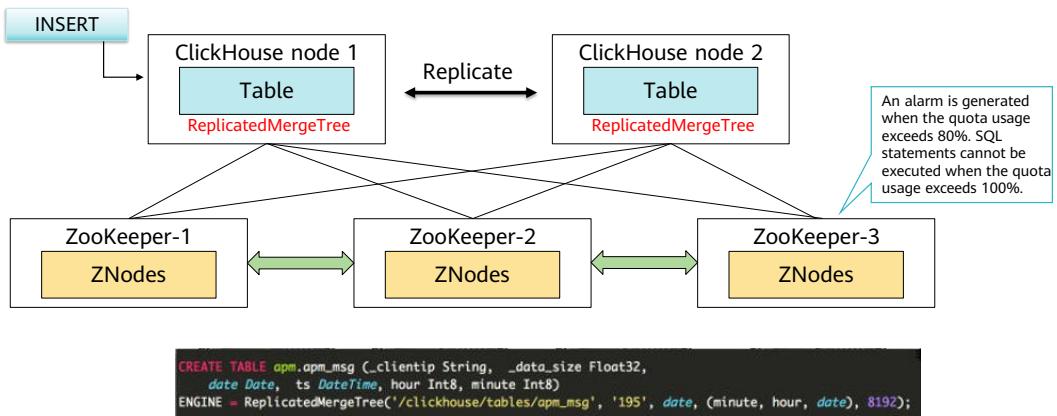
- Online scale-out and visualized data migration are supported. Data in a single table can be migrated by node, rate limiting is supported, and the migration progress and results can be viewed.

- Security hardening, Kerberos-based user authentication, and compatibility with the community
- Visualized permission management, no need to perform command line interface (CLI) operations in the background; read and write permission control at the database and table levels

Rolling Upgrade/Restart

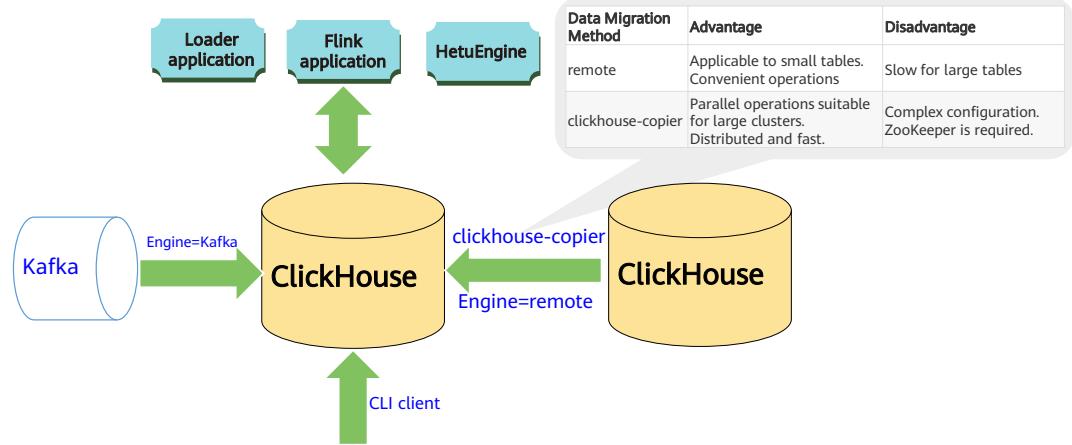


Automatic Cluster Topology and ZooKeeper Overload Prevention



- The two-copy mode is used by default. In this mode, the cluster shard topology is automatically generated, ZooKeeper overload prevention is enabled, and the threshold alarm function is enabled.

Peripheral Ecosystem Interconnection



Quiz

1. (Single-choice) ClickHouse is a column-store database applicable to the () scenario.
 - A. OLTP
 - B. OLAP
2. (Multiple-choice) Which of the following are ClickHouse table engines? ()
 - A. MergeTree
 - B. ReplicatedMergeTree
 - C. Tree
 - D. DistributedTree

- Answers:

1. B
2. ABD

Summary

- This chapter describes ClickHouse, an OLAP DBMS, including its overview, use cases, system architecture, basic features, and enhanced features.

Acronyms and Abbreviations

- OLAP: Online Analytical Processing
- DBMS: Database Management System
- OLTP: Online Transaction Processing
- BI: Business Intelligence
- CLI: Command Line Interface

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive
statements including, without limitation, statements regarding
the future financial and operating results, future product
portfolio, new technology, etc. There are a number of factors
that could cause actual results and developments to differ
materially from those expressed or implied in the predictive
statements. Therefore, such information is provided for reference
purpose only and constitutes neither an offer nor an acceptance.
Huawei may change the information at any time without notice.



MapReduce and YARN Technical Principles



Foreword

- This course describes MapReduce and YARN.
- MapReduce is the most famous computing framework for batch and offline processing in the big data field. You will learn its principles, processes, and use cases.
- YARN is a component responsible for centralized resource management and scheduling in the Hadoop cluster. You will learn its definition, functions and architecture, HA solution, fault tolerance mechanism, and usage to allocate resources.
- We will also briefly introduce the enhanced features that Huawei provides for these components.

Objectives

- Upon completion of this course, you will be able to:
 - Be familiar with the concepts of MapReduce and YARN.
 - Master the use cases and principles of MapReduce.
 - Master the functions and architectures of MapReduce and YARN.
 - Be familiar with YARN features.

Contents

- 1. MapReduce and YARN Overview**
2. Functions and Architectures of MapReduce and YARN
3. Resource Management and Task Scheduling in YARN
4. Enhanced Features of YARN

MapReduce Overview

- MapReduce is designed and developed based on the MapReduce paper published by Google. MapReduce is used for parallel and offline computing of large-scale data sets (above 1 TB).
- Features
 - Highly abstract programming idea: Programmers only need to describe what to do, and the execution framework will process the work accordingly.
 - Outstanding scalability: Cluster capabilities can be improved by adding nodes.
 - High fault tolerance: Cluster availability and fault tolerance are improved using policies such as computing or data migration.

- When a node is faulty, tasks running on the faulty node can be performed on another node using computing or data migration.
 - MapReduce is a cluster-based high-performance parallel computing platform (cluster infrastructure).
 - MapReduce is a parallel computing and running software framework.
 - MapReduce is a parallel programming model and methodology.

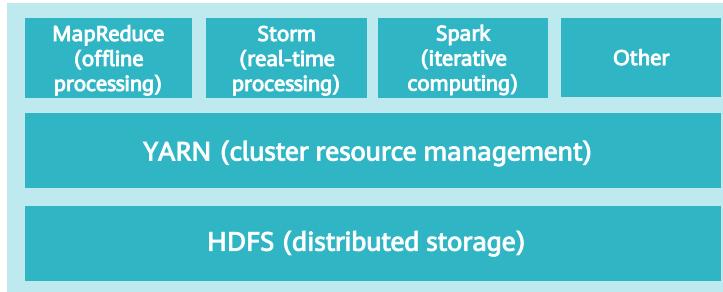
Resource Scheduling and Allocation

- Hadoop 1.0 only has HDFS and MapReduce. Resources are scheduled using MapReduce 1.0, which has the following disadvantages:
 - The master node has SPOFs. Fault recovery depends on the periodic checkpoint, which does not ensure the reliability. When a fault occurs, the system notifies users who have to manually decide whether to perform recalculation.
 - Job and resource scheduling are not distinguished. When MapReduce is running, many concurrent jobs exist in the environment. To solve this, diversified and efficient scheduling policies are needed.
 - When many jobs are concurrently executed without considering resource isolation or security, it is difficult to ensure that a single job does not occupy too many resources and that user programs are secure.
- Hadoop 2.0 introduces the YARN framework to better schedule and allocate cluster resources to overcome the shortcomings of Hadoop 1.0 and support various programming requirements.

- As big data analysis evolves, we need a wider range of distributed programming paradigms, such as real-time data processing, in-memory computing, and graph computing. This poses new challenges to the runtime environment, which needs to be more universal to support different programming models beyond the MapReduce programming paradigm like the MapReduce framework. Different programming paradigms or computing tasks have different requirements on resources (such as CPU and memory). So, we need excellent scheduling policies to maximize resource utilization while delivering what each application needs. In addition, tasks need to be isolated to avoid mutual impact.

YARN Overview

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



Contents

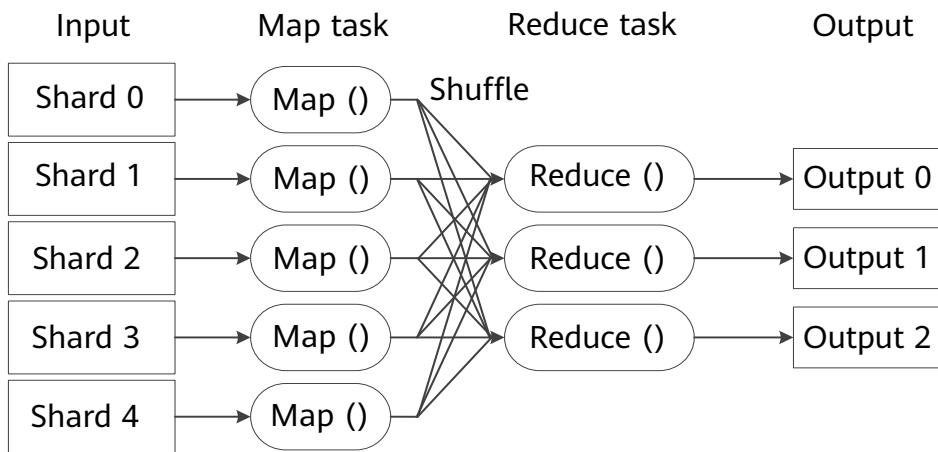
1. MapReduce and YARN Overview
- 2. Functions and Architectures of MapReduce and YARN**
3. Resource Management and Task Scheduling in YARN
4. Enhanced Features of YARN

MapReduce Process

- 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. Then, further processing may be performed to obtain a result.

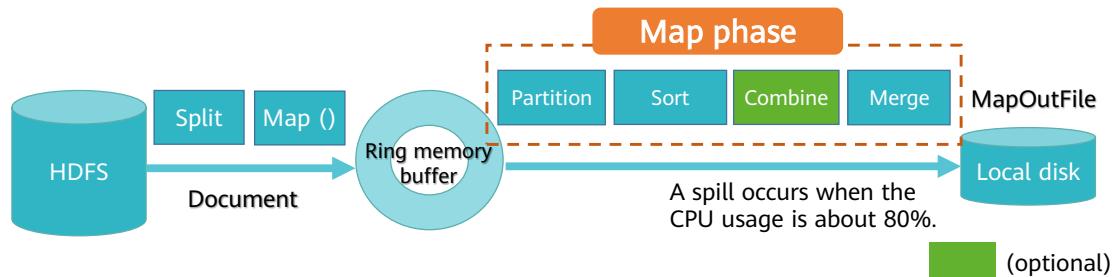
- MapReduce is inspired by the built-in functions **map** and **reduce** in functional languages (such as Lisp).
 - In a functional language, **map** indicates that each element in a list is calculated, and **reduce** indicates that each element in a list is calculated iteratively. The specific calculation is implemented through the input functions. The **map** and **reduce** functions provide the calculation framework. But it is too far from this explanation to the real-world MapReduce, and it still takes a leap. If **reduce** can perform iterative calculation, the elements in the list are related. For example, if you want to sum all elements, they must at least be numbers. However, the **map** processes each element in the list separately, which indicates that the data in the list can be disordered. So, it is a little connected.

MapReduce Working Process



Map Phase

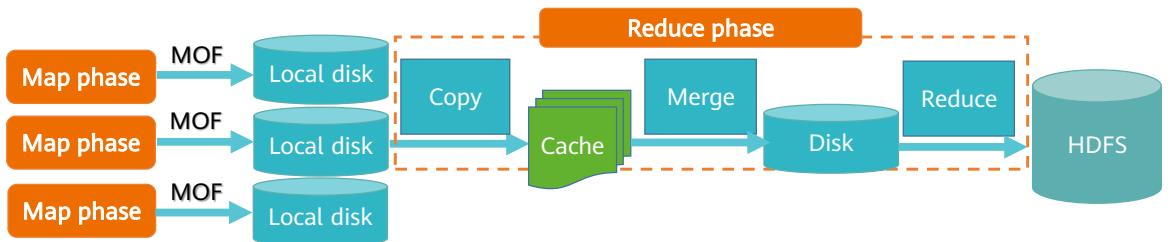
- Before jobs are submitted, the files to be processed are split. By default, the MapReduce framework regards a block as a split. Client applications can redefine the mapping between blocks and splits.
- In the Map phase, data is stored in a ring memory buffer. When the data in the buffer reaches about 80%, a 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 Map outputs are sorted, for example, ('Hi','1'), ('Hello','1') is reordered as ('Hello','1'),('Hi','1').
 - By default, combination is optional in the MR framework. For example, you can combine ('Hi', '1'), ('Hi', '1'), ('Hello', '1'), and ('Hello', '1') into ('Hi', '2'). ('Hello', '2').
 - Spill: After a Map task is processed, many spill files are generated. These spill files must be combined into spill files (MapOutFile, MOF) that has been partitioned and sorted. To reduce the amount of data to be written to disks, MapReduce allows MOFs to be written after being compressed.
- The differences between Combine and Merge:
 - If the two key-value pairs <"a",1> and <"a",1> are combined, <"a",2> is obtained; if the two key value pairs are merged, the result is <"a",<1,1>.

Reduce Phase

- MOF files need to be sorted. If Reduce tasks receive little data, 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 one large file. Many intermediate files are generated during the merge. The last merge result is directly output to the Reduce function defined by the user.
- A small volume of data does not need to be spilled to the disk. Instead, the data is merged in the cache and then output to Reduce.



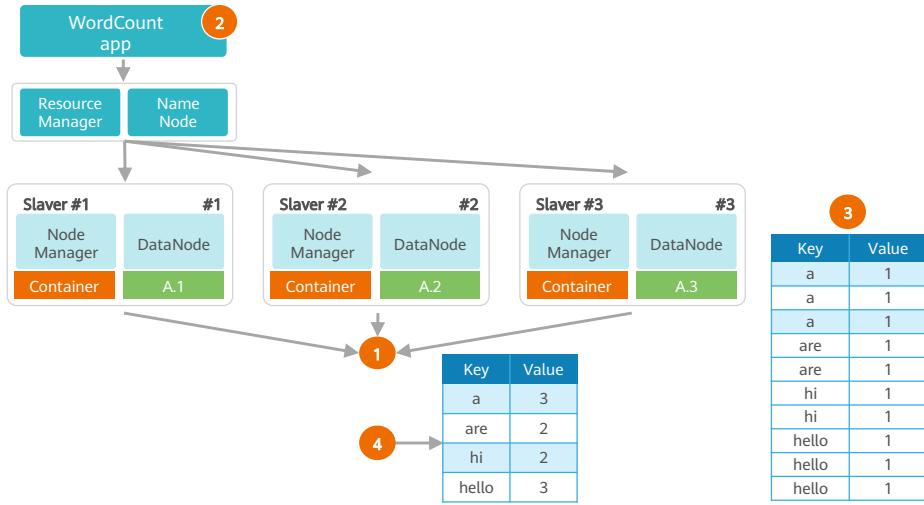
- Typically, when the MOF output progress of a Map task reaches 3%, a Reduce task is started to obtain MOF files from each Map task. The number of Reduce tasks is decided by clients and determines the number of MOF partitions. As such, the MOF files output by Map tasks map to Reduce tasks.

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 as well as sorting and merging MOF files.

- A cache is allocated to each Map task. By default, MapReduce has 100 MB of cache. Set the spill ratio to 0.8. Sorting is a default operation. After sorting, you can combine the target data.
- Before all Map tasks are completed, the system merges the files into a large file and saves the file on the local disk.
- During the merge, if there are more spill files 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 asks the JobTracker whether the Map task is completed through a remote procedure call (RPC). If it is, the Reduce task obtains data.
- The obtained data is stored to the cache, 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.

Example WordCount

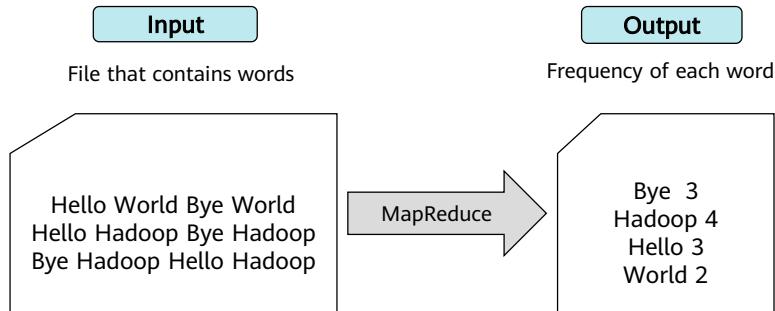


14 Huawei Confidential

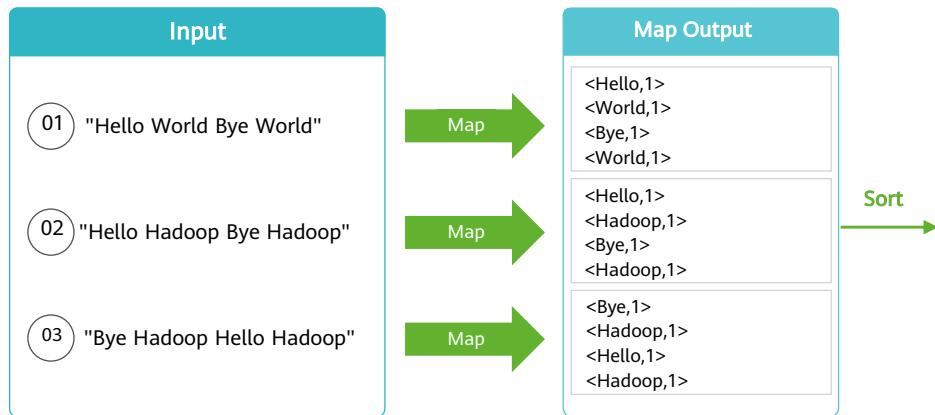


- If a user needs to analyze the frequency of each word in text file A, the MapReduce framework can quickly help with the statistical analysis.
 - Step 1: File A is stored on HDFS and divided into blocks A.1, A.2, and A.3 that are stored on DataNodes #1, #2, and #3, respectively.
 - Step 2: The WordCount 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 creates three Map tasks as well as three Reduce tasks that run 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 displayed table on the slide.
 - 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 on the right are output.

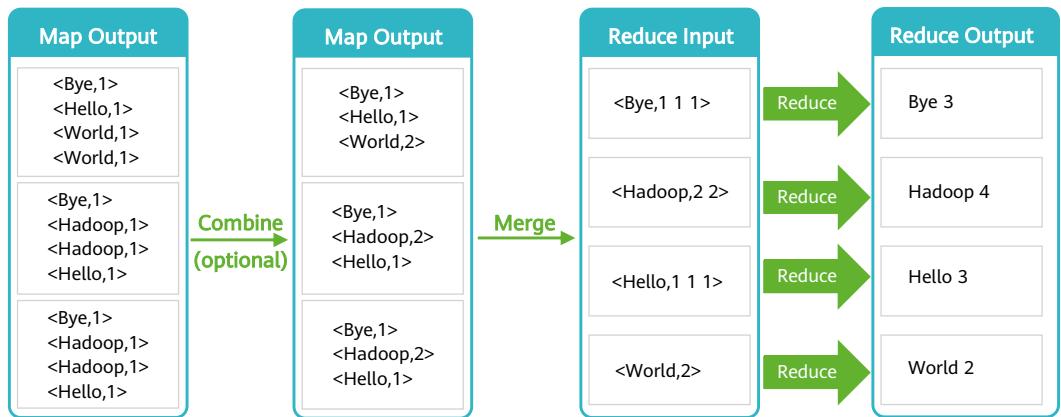
Functions of WordCount



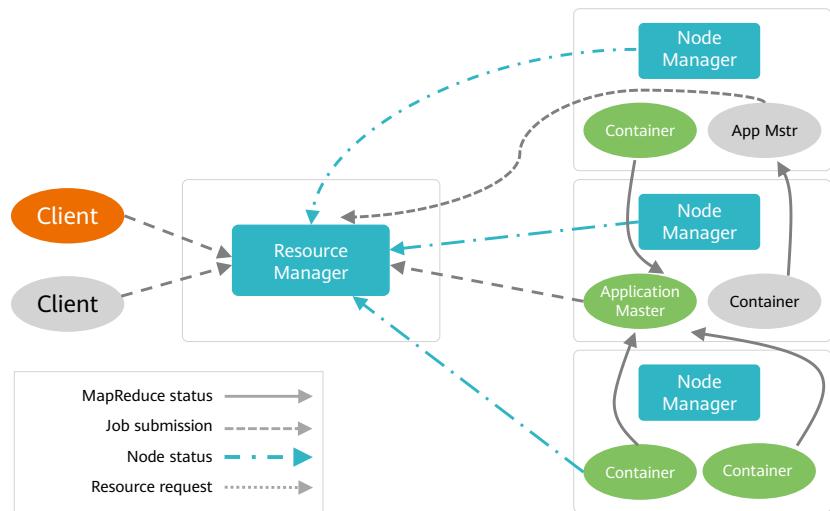
Map Process of WordCount



Reduce Process of WordCount



YARN Architecture

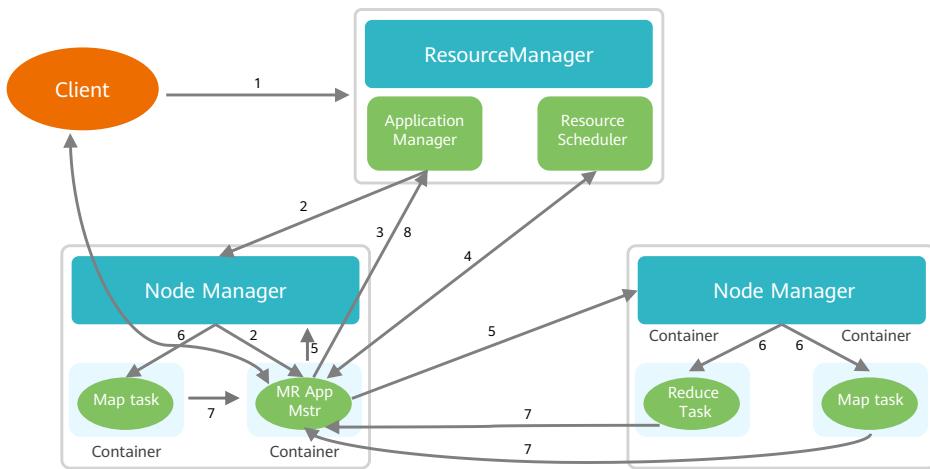


18 Huawei Confidential

HUAWEI

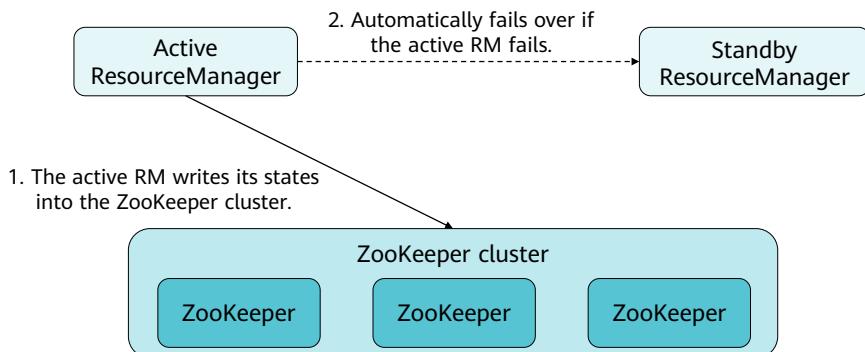
- ResourceManager
 - Centrally manages and allocates all resources in the cluster.
- ApplicationMaster
 - Applies for and releases resources from the scheduler, requests NodeManager to run tasks, traces application status, and monitors their processes.
- NodeManager
 - Interacts with ApplicationMaster and ResourceManager.
 - Receives and executes container-related commands, and reports the container health status.
- Container
 - Encapsulates a certain number of resources on a node.
- In the figure, two clients submit jobs to YARN. The client in blue indicates one job process, and the client in gray indicates another job process.
- The clients submit jobs. ResourceManager receives the jobs and starts and monitors the first container, that is, ApplicationMaster.
- App Mstr notifies NodeManager to manage resources and start other containers.
- The jobs are running in containers.

Job Scheduling Process of MapReduce on YARN



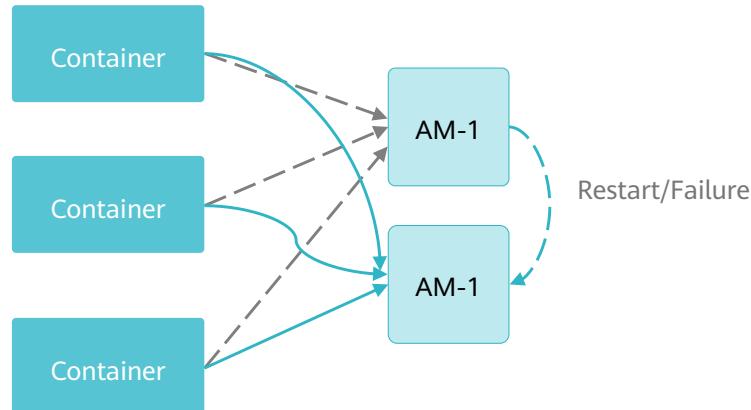
HA Solution of YARN

- 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, the active-standby switchover can be triggered automatically or manually.
- When the automatic failover function is not enabled, after the YARN cluster is enabled, administrators need to run the **yarn rmadmin** command to manually switch one of the ResourceManager nodes to the active state. Upon planned maintenance or a fault, they should first demote the active ResourceManager to standby and then promote the standby ResourceManager to active.
- When automatic switchover is enabled, a built-in ActiveStandbyElector that is based on ZooKeeper decides which ResourceManager node should be the active one. When the active ResourceManager is faulty, another ResourceManager node is automatically selected to take over.
- When ResourceManager nodes in the cluster are deployed in HA mode, the configuration **yarn-site.xml** used by clients needs to list all the ResourceManager nodes. Clients (including ApplicationMaster and NodeManager) try connecting to the ResourceManager nodes in polling mode until they hit the active ResourceManager node. If it cannot connect to the active ResourceManager, the client continues searching for a new one in polling mode.

Fault Tolerance Mechanism of YARN ApplicationMaster



- In YARN, ApplicationMaster runs on NodeManager just like other containers (ApplicationMaster that is not managed is ignored for this context). ApplicationMaster may break down, exit, or shut down. If an ApplicationMaster node goes down, ResourceManager kills all the containers of that ApplicationAttempt. This includes any containers running on NodeManager. ResourceManager starts a new ApplicationAttempt node on another compute node.
- Different types of applications want to handle ApplicationMaster restart events differently. The objective of MapReduce applications is not to lose the task status, but some status loss is allowed. 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 without faults.

Contents

1. MapReduce and YARN Overview
2. Functions and Architectures of MapReduce and YARN
- 3. Resource Management and Task Scheduling in YARN**
4. Enhanced Features of YARN

Resource Management

- Available memory and CPU resources allocated to each NodeManager can be configured through the following parameters (on the YARN service configuration page):
 - **yarn.nodemanager.resource.memory-mb**: size of physical memory that YARN can allocate to a container
 - **yarn.nodemanager.vmem-pmem-ratio**: ratio of the virtual memory to the physical memory
 - **yarn.nodemanager.resource.cpu-vcore**: number of vCores that can be allocated to a container
- In Hadoop 3.x, the YARN resource model can support user-defined countable resource types, not just CPU and memory.
 - Common countable resource types include GPU resources, software licenses, and locally-attached storage, but do not include ports and labels.

- **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.

YARN Resource Schedulers

- In YARN, a scheduler allocates resources to applications; three schedulers are available based on different policies:
 - FIFO scheduler: Applications are arranged into a first-in-first-out (FIFO) queue based on the submission sequence. Resources are first allocated to the application at the top of the queue. Once the first application meets the requirements, resources are allocated to the next application, and so on.
 - Capacity scheduler: Multiple organizations are allowed to share the entire cluster. Each organization can obtain some of the cluster's computing capabilities. A dedicated FIFO 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 to be shared by multiple members in an organization.
 - Fairs scheduler: All applications have an equal share of resources. (The fairness can be configured.)

- Ideally, the requests for Yarn resources should be met immediately. However, in reality, resources are limited. Especially in a busy cluster, it usually takes some time for an application to obtain resources. In YARN, a scheduler allocates resources to applications. In fact, scheduling is a difficult issue because it is challenging to find an appropriate policy to solve resource scheduling in all use cases. So, YARN provides multiple schedulers and configurable policies.
- For example, there are two users, user A and user B, who use Fair Scheduler, and both have a queue. If A starts a job but B does not run any jobs, A obtains all cluster resources. After B starts a job, A's job continues to run. However, after a while, each of the two jobs obtains half of the cluster resources. If B starts a second job and the other jobs are still running, the second job shares the resources of B's queue with the first job. That is, each job of B gets a quarter of the total cluster resources, and A's job still has half. Eventually, the resources are shared equally between the two users.

Capacity Scheduler Overview

- Capacity scheduler enables Hadoop applications to run in a shared, multi-tenant cluster while maximizing its throughput and utilization of the cluster.
- Capacity scheduler 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 the 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, the administrator can automatically configure and manage the queue in the **administer_queue ACL** of the queue.

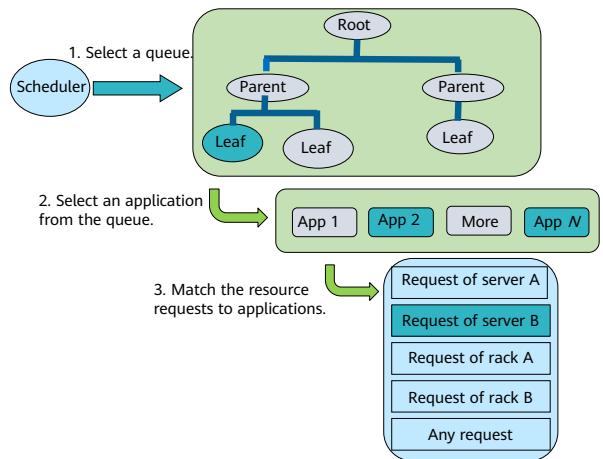
25 Huawei Confidential



- Capacity scheduler

Resource Allocation Model

- A 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 the 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.

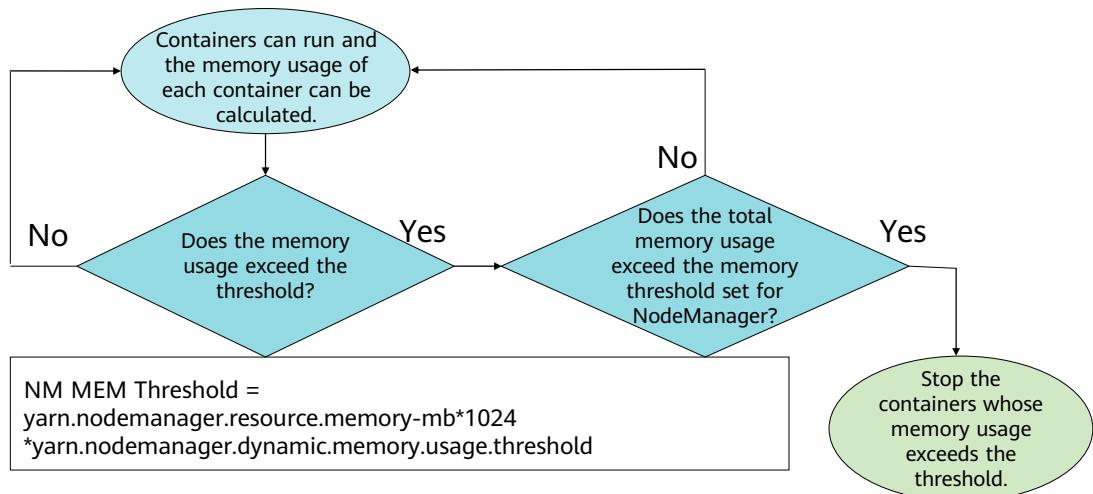


- Capacity scheduler

Contents

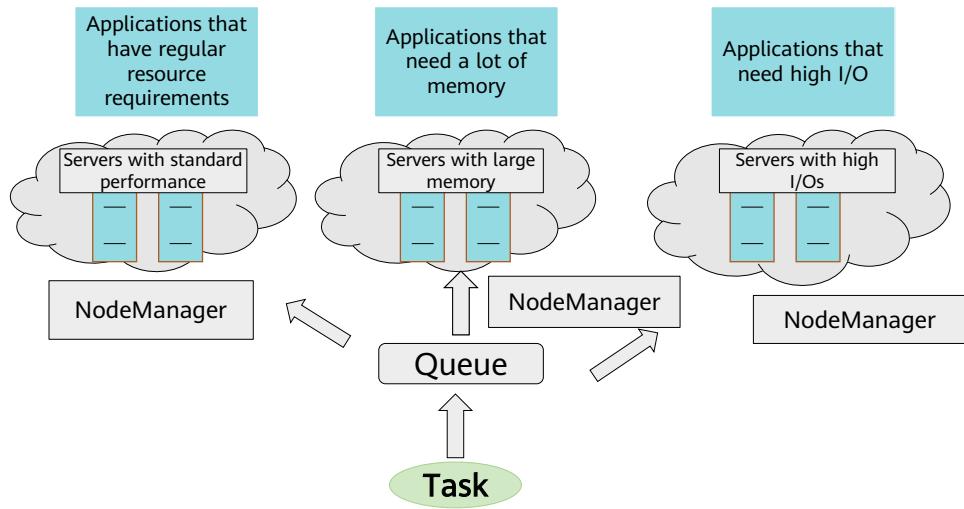
1. MapReduce and YARN Overview
2. Functions and Architectures of MapReduce and YARN
3. Resource Management and Task Scheduling in YARN
- 4. Enhanced Features of YARN**

Enhanced Feature — Dynamic YARN Memory Management



- Dynamic memory management can be used to optimize the memory usage of containers in NodeManager. Multiple containers may be generated when tasks are running.
- Currently, 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 often causes job failures.
- Dynamic memory management addresses this issue. Containers that use excessive memory are terminated only when the total memory usage of all containers in NodeManager exceeds the specified threshold. The threshold (unit: GB) is calculated using the following formula: **yarn.nodemanager.resource.memory-mb x 1024 x yarn.nodemanager.dynamic.memory.usage.threshold**.
- For example, if the physical memory usage of some containers exceeds the configured memory threshold but the total memory does not, then the overused containers can continue running.

Enhanced Feature — Label-based YARN Scheduling



- Before label-based scheduling, it was 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.
- Label-based scheduling is a scheduling policy that enables users to label each NodeManager, for example, labeling NodeManager with high-memory or high-IO. Users can also 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 run on labeled nodes.
- For example, memory-intensive tasks submitted to queues with high-memory labels can run on servers with large memory.

Quiz

1. (Multiple-choice) What are the highlights of MapReduce? ()
 - A. Easy programming
 - B. Outstanding scalability
 - C. Real-time computing
 - D. High fault tolerance

- Answer: ABD

Quiz

2. (Single-choice) What is an abstraction of YARN resources? ()

- A. Memory
- B. CPU
- C. Container
- D. Disk space

- Answer: C

Quiz

3. (Single-choice) What does MapReduce apply to? ()
 - A. Iterative computing
 - B. Offline computing
 - C. Real-time interactive computing
 - D. Stream computing

- Answer: B

Summary

- This chapter describes the use cases and basic architectures of MapReduce and YARN, principles of YARN resource management and task scheduling, and enhanced features of YARN in Huawei MRS clusters.

Acronyms and Abbreviations

- MOF: Map Out File, which is an output file in the Map phase
- FIFO: First Input First Output
- ACL: Access Control List

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



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



Foreword

- This chapter describes Spark and Flink. Spark is an in-memory distributed computing engine, and Flink implements stream and batch processing in a single engine. The first part describes the basic concepts of Spark and the similarities and differences between the Resilient Distributed Dataset (RDD), DataSet, and DataFrame data structures in Spark. It also covers the features of Spark SQL, Spark Streaming, and Structured Streaming. The second part describes the basic concepts, principles, architecture, time, window, watermark, and fault tolerance mechanism of Flink.

Objectives

- Upon completion of this course, you will be able to:
 - Understand the use cases and highlights of Spark.
 - Master Spark programming and technical framework.
 - Master the core technologies and architecture of Flink.
 - Master the time and window mechanism of Flink.
 - Master the fault tolerance mechanism of Flink.

Contents

- 1. Spark — In-memory Distributed Computing Engine**
 - **Spark Overview**
 - **Spark Data Structure**
 - **Spark Principles and Architecture**
- 2. Flink — Stream and Batch Processing in a Single Engine**

Spark Overview

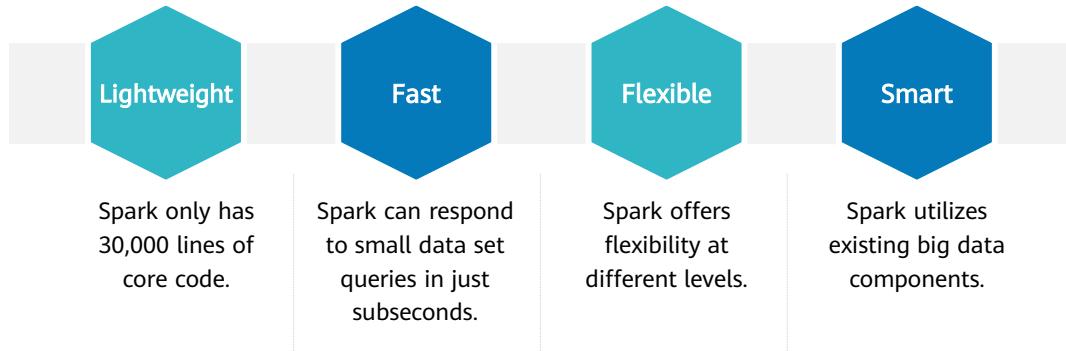
- Apache Spark started as a research project at the UC Berkeley AMPLab in 2009. After being released, Spark grew into a broad developer community, and moved to the Apache Software Foundation in 2013. Today, the project is developed collaboratively by a community of hundreds of developers from hundreds of organizations.
- It is a fast, versatile, and scalable memory-based big data computing engine.
- It is a one-stop solution that integrates batch processing, real-time streaming, interactive query, graph programming, and machine learning.

Spark 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.
- Streaming processing can be used for real-time businesses analysis (such as page-click streams), recommendation systems, and public opinion analysis.

- Users benefit more from Spark when they need to frequently perform certain operations and there is a large data volume because it is already stored in memory.
- Public opinion analysis: Monitors, analyzes, and processes sensitive and trending Internet information.

Highlights of Spark



- **Lightweight**: Spark only has 30,000 lines of core code.
 - Spark is developed in the easy-to-read and rich Scala language.
 - It smartly leverages the infrastructure of Hadoop and Mesos.
- **Fast**: Spark can respond to small data set queries in just subseconds.
 - Spark is higher 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 utilizes existing big data components.
 - Spark seamlessly combines with Hadoop.
 - Graph computing uses Pregel and PowerGraph APIs as well as the point division

idea of PowerGraph.

Spark vs. MapReduce

	Hadoop	Spark	Spark
Data Size	102.5 TB	102 TB	1000 TB
Time Required (Minutes)	72	23	234
Number of Nodes	2100	206	190
Cores	50400	6592	6080
Rate	1.4 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Daytona Gray Sort	Yes	Yes	Yes

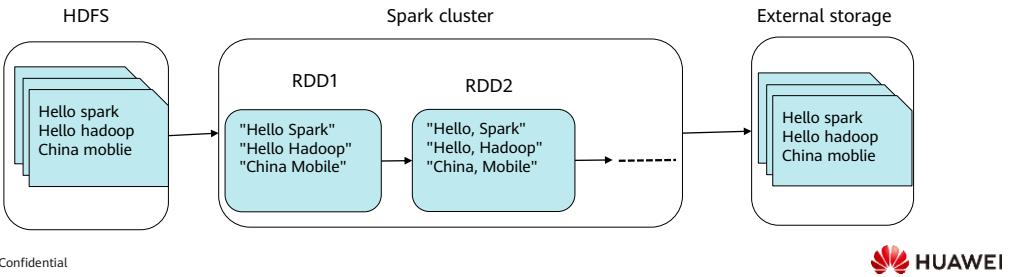
- Spark uses one-tenth of the resources used by MapReduce but provides three times better performance.
- Cores: indicates the total number of cluster cores.
- Rate: indicates the data read speed of a cluster.
- Rate/node: indicates the average data read speed per node.
- Daytona Gray: indicates a common sort competition of Sort Benchmark.

Contents

- 1. Spark — In-memory Distributed Computing Engine**
 - Spark Overview
 - **Spark Data Structure**
 - Spark Principles and Architecture
- 2. Flink — Stream and Batch Processing in a Single Engine**

RDD — Core Concept of Spark

- RDDs are elastic, read-only, and partitioned distributed datasets.
- By default, RDDs are stored in the memory and are written to disks when memory is insufficient.
- RDD data is stored in clusters as partitions.
- RDDs have a lineage mechanism, which allows for rapid data recovery when data is lost.

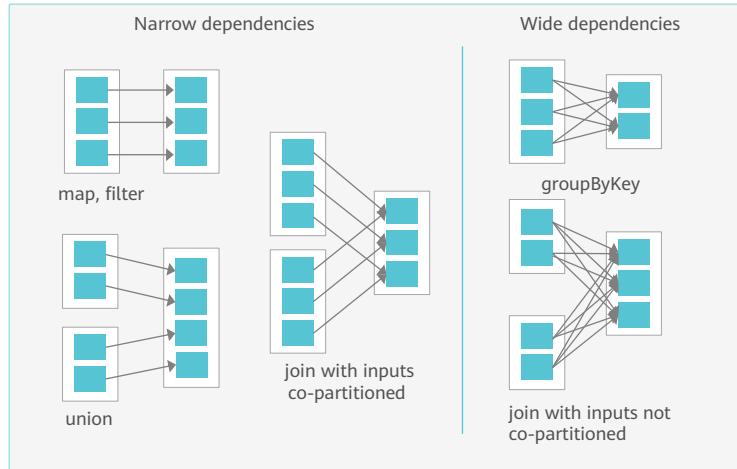


10 Huawei Confidential

HUAWEI

- An RDD is the abstraction of basic data in Spark.
- RDD generation: An RDD can be created from a Hadoop file system or other storage systems that are compatible with Hadoop, such as Hadoop Distributed File System (HDFS). Alternatively, an RDD can be 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.
- RDD partition: 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 datasets.
- RDD advantages: RDDs are read-only and static. Therefore, Spark can provide a higher fault tolerance and implement speculative execution.

RDD Dependencies

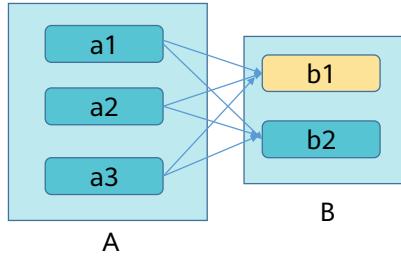


Differences Between Wide and Narrow Dependencies — Operator

- A narrow dependency indicates that only one child RDD can use each partition of a parent RDD, for example, **map**, **filter**, and **union**.
- A wide dependency indicates that the partitions of multiple child RDDs depend on the partition of the same parent RDD, for example, **groupByKey**, **reduceByKey**, and **sortByKey**.

Differences Between Wide and Narrow Dependencies — Fault Tolerance

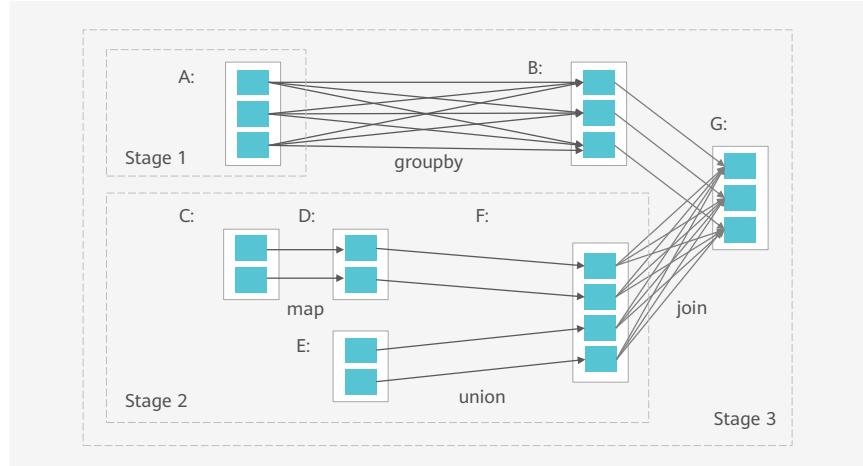
- If a node is faulty:
 - Narrow dependency: Only the parent RDD partition corresponding to the child RDD partition needs to be recalculated.
 - Wide dependency: In extreme cases, all parent RDD partitions need to be recalculated.
- As shown in the following figure, if the b1 partition is lost, a1, a2, and a3 need to be recalculated.



Differences Between Wide and Narrow Dependencies — Data Transmission

- Wide dependency usually corresponds to Shuffle operations. During running, the partition of the same parent RDD needs to be transferred to different child RDD partitions, which 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.

Stage Division of RDD



RDD Operation Type

- Spark operations can be classified into creation, conversion, control, and behavior.
 - Creation: used to create an RDD. An RDD is created through a memory collection and an external storage system, or by a transformation operation.
 - Transformation: An 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.
 - Control: RDD persistence is performed. An 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.
 - Action: an operation that can trigger Spark running. There are two types of action operations in Spark. One is to output the calculation result, and the other is to save the RDD to an external file system or database.

- The transformation operation is not executed immediately. Instead, Spark records the information about the operation to be executed. The transformed RDD will not be computed until an action is called.

Creation Operation

- Currently, there are two types of basic RDD
 - Parallel collection: An existing collection is collected and computed in parallel.
 - External storage: A function is executed on each record in a file. The file system must be the HDFS or any storage system supported by Hadoop.
- 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 Operation

- Spark can persistently store RDD to the memory or disk file system persistently. The RDD in memory can significantly improve iterative computing and data sharing between computing models. Generally, 60% of the memory of the execution node is used to cache data, and the remaining 40% is used to run tasks. In Spark, **persist** and **cache** operations are used for data persistency. Cache is a special case of **persist()**.

Transformation Operation — Transformation Operator

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

Action Operation — Action Operator

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 an RDD.
first()	Obtains the first element of a dataset.
take(n)	Obtains the top elements of a dataset and returns an array.
saveAsTextFile(path)	This API is used to write the dataset to a text file or HDFS. Spark converts each record into a row and writes the row to the file.

DataFrame

- Similar to RDD, DataFrame is also an invariable, elastic, and distributed dataset. It records the data structure information, that is, schema, in addition to data. The schema is similar to a two-dimensional table.
- The query plan of DataFrame can be optimized by Spark Catalyst Optimizer. Spark is easy to use, which improves high-quality execution efficiency. Programs written by DataFrame can be converted into efficient forms for execution.

- **Catalyst working process:** SQL statements are parsed into a syntax tree by the Parser module. The syntax tree is called Unresolved Logical Plan, which is parsed into a logical plan by the Analyzer module with the help of metadata. In this case, users should perform in-depth optimization based on various rule-oriented optimization policies to obtain the optimized logical plan. The optimized logical execution plan is still logical and cannot be understood by the Spark system. So, it needs to be converted into a physical plan.

DataSet

- 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.
- DataSet, a typed dataset, includes Dataset[Car] and Dataset[Person].

Differences Between DataFrame, DataSet, and RDD

- Assume that there are two lines of data in an RDD, which are displayed as follows:

```
1, Lucy, 23  
2, Tony, 35
```

- The data in DataFrame is displayed as follows:

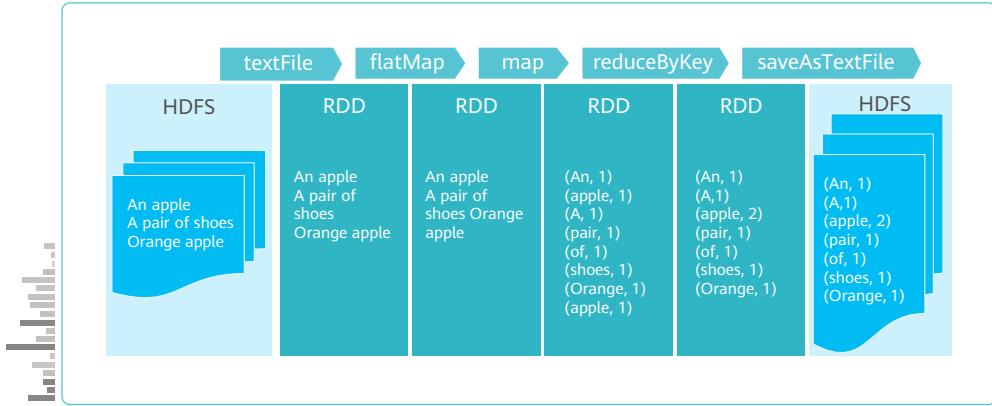
ID:String	Name:String	Age:int
1	Lucy	23
2	Tony	35

- The data in DataSet is displayed as follows:

```
Value:People[age:bright,id:bigint,name:string]  
People(id=1,name="Lucy",age=23)  
People(id=2,name="Tony",age=35)
```

- Differences between DataFrame and DataSet
 - DataFrame
 - The type of each row in DataFrame is fixed to Row. The value of each field can be obtained only after parsing, and the value of each column cannot be directly accessed.
 - The DataFrame compiler lacks security check for types.
 - DataSet
 - The type of each row is not fixed. It can be Person or Row.
 - The DataSet type is more secure.
- Differences between RDD and DataFrame/DataSet
 - RDD
 - Used for APIs of Spark1.X modules.
 - The SparkSQL operation is not supported.
 - Automatic code optimization is not supported.
 - Difference between DataFrame and DataSet
 - Used for APIs of Spark2.X modules.
 - Supports SparkSQL operations. Temporary tables can be registered, where SQL statements can be executed.
 - Supports some convenient storage, for example, saving files in CSV or JSON format.
 - Supports automatic code optimization because they are built based on the SparkSQL engine.

Typical Case — WordCount



WordCount

```
object WordCount
{
    def main (args: Array[String]): Unit = {
        // Configure the Spark application name.
        val conf = new
        SparkConf().setAppName("WordCount")
        val sc: SparkContext = new SparkContext(conf)
        val textFile = sc.textFile("hdfs://...")
        val counts = textFile.flatMap(line => line.split(" "))
            .map(word => (word, 1))
            .reduceByKey(_ + _)
        counts.saveAsTextFile("hdfs://...")
    }
}
```

Create a SparkContext object and set the application name to **Wordcount**.

Invoke the transformation of the RDD for calculation. Split the text file by space, set the count of each word to 1, and combine the counts in the same key. This step is distributed to each Executor for execution.

Load the text file from HDFS to obtain the RDD data set.

Invoke the action to save the result. This line triggers the actual task execution.

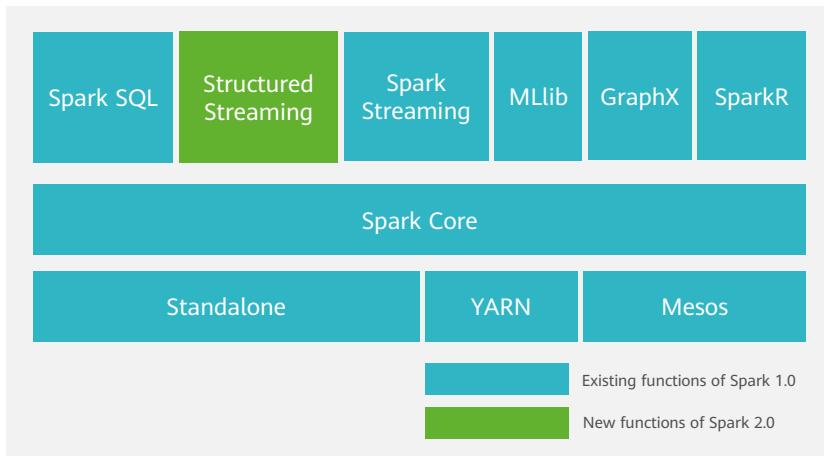
Contents

1. Spark — In-memory Distributed Computing Engine

- Spark Overview
- Spark Data Structure
- Spark Principles and Architecture

2. Flink — Stream and Batch Processing in a Single Engine

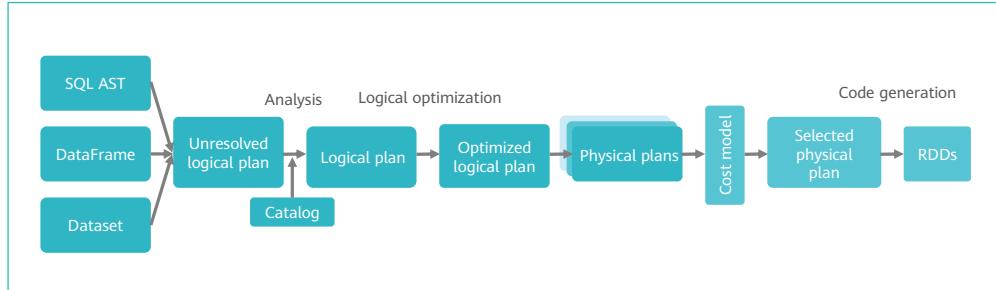
Spark Architecture



- **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 only the Spark On Yarn mode. Other modes are not currently 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.
- **MLlib and GraphX:** are algorithm libraries.
- **Structured Streaming:** is unique to Spark 2.0 or later.

Spark SQL Overview

- Spark SQL is a module used for structured data processing in Spark. In Spark applications, you can seamlessly use SQL statements or DataFrame APIs to query structured data.



Spark SQL vs. Hive

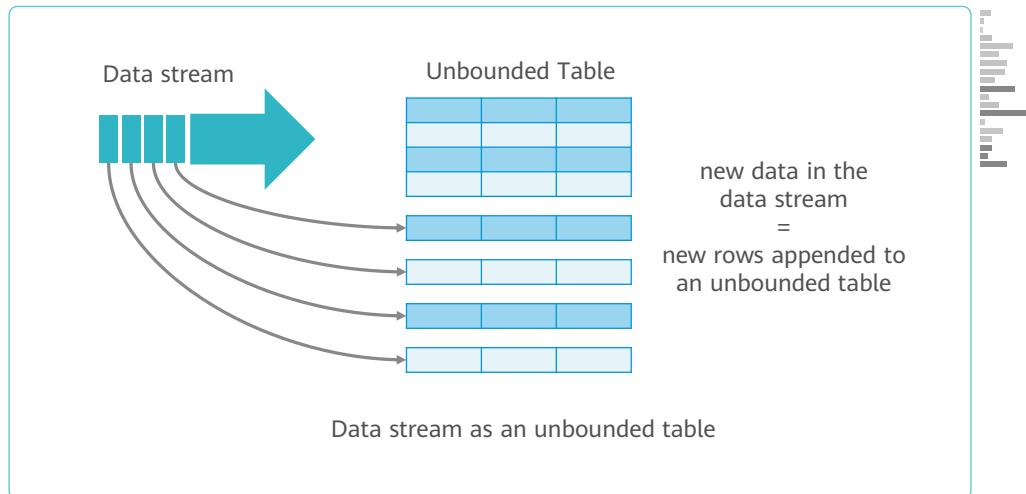
- Differences
 - The execution engine of Spark SQL is Spark Core, and the default execution engine of Hive is MapReduce.
 - Spark SQL executes 10 to 100 times faster than Hive.
 - Spark SQL does not support buckets, but Hive does.
- Relationships
 - Spark SQL depends on Hive's metadata.
 - Spark SQL is compatible with most Hive syntax and functions.
 - Spark SQL can use the custom functions of Hive.

- Spark SQL syntax and Hive syntax are basically the same except for bucket operations.
- Spark SQL is compatible with Hive functions.

Structured Streaming Overview

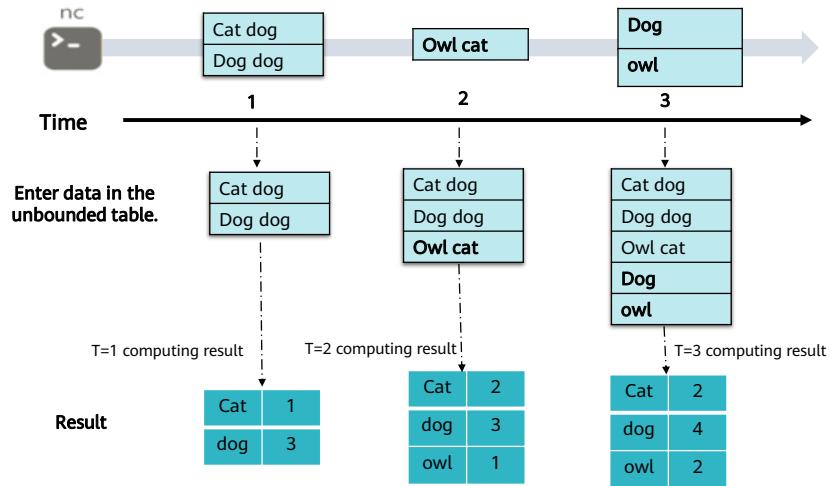
- Structured Streaming is a stream processing engine built on the Spark SQL engine. You can compile a streaming computing process like using static RDD data. Spark SQL will take care of running it incrementally and continuously, and then updates the final result as streaming data continues to arrive.

Structured Streaming Overview



- 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.
- Unbounded table: New data is coming in and the old data is discarded. This is actually a continuous structured data streaming.

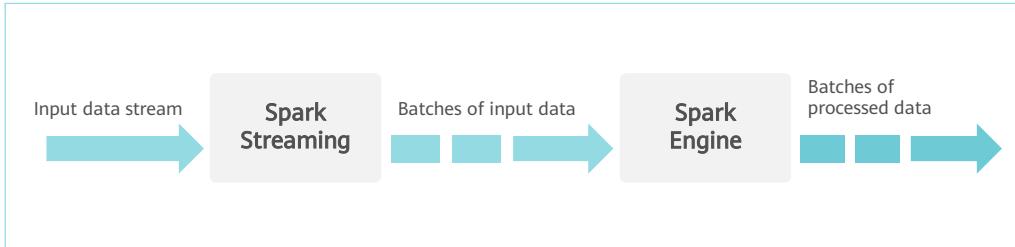
Sample Programming Model of Structured Streaming



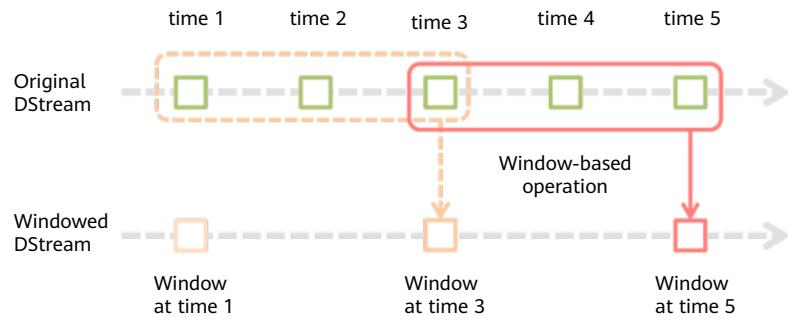
- The first lines in a DataFrame object are an input table with data input, and the last WordCounts DataFrame is a result table with the result set. The query on the lines in the DataFrame data stream generates the wordCounts expression, which is the same as the usage on a static DataFrame. However, Spark monitors socket connections to obtain new and continuously generated data. When new data is generated, Spark runs an incremental **counts** query on the new data, integrates the new and previously calculated **counts**, and obtains updated **counts**.

Spark Streaming Overview

- The basic principle of Spark Streaming is to split real-time input data streams by time slice (in seconds), and then use the Spark engine to process data of each time slice in a way similar to batch processing.



Window Interval and Sliding Interval



A window slides on DStream, merges and performs operations on the RDDs that fall into the window, and generates windowed RDDs.

Window length: indicates the duration of a window.

Sliding window interval: indicates the interval for performing window operations.

Spark Streaming vs. Storm

Parameter	Storm	Spark Streaming
Real-time computing model	Real time: A piece of data is processed once it is requested.	Quasi real time: Data within a time range is collected and processed as an RDD.
Real-time computing delay	In milliseconds	In seconds
Throughput	Low	High
Transaction mechanism	Supported and perfect	Supported but not perfect
Fault tolerance	High for ZooKeeper and Acker	Normal for Checkpoint and WAL
Dynamic adjustment parallelism	Supported	Not supported

- Spark Streaming and Storm are both excellent at real-time computing, but serve different specific scenarios. Spark Streaming is only superior to Storm in throughput.
- 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.
- Spark Streaming:
 - 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.

Quiz

1. What are the features of Spark?
2. What are the advantages of Spark over MapReduce?
3. What are the differences between wide and narrow dependencies of Spark?
4. What are the use cases of Spark?

- 1. Lightweight, fast, flexible, and smart
- 2. In-memory fast computing
- 3. A narrow dependency indicates that only one child RDD can use each partition of a parent RDD. A wide dependency indicates that the partitions of multiple child RDDs depend on the partition of the same parent RDD.
- 4. Offline batch processing, real-time stream processing, and interactive query

Quiz

5. RDD operators are classified into:__and__.
6. The__module is the core module of Spark.
7. RDD dependency types include__and__.

- 1. Transformation, action
- 2. Spark core
- 3. Wide dependency, narrow dependency

Contents

1. Spark — In-memory Distributed Computing Engine
2. **Flink — Stream and Batch Processing in a Single Engine**
 - Flink Principles and Architecture
 - Flink Time and Window
 - Flink Watermark
 - Flink Fault Tolerance Mechanism

Flink Overview

- Flink started through joint research by Berlin University of Technology, Humboldt University of Berlin, and Hasso Plattner Institute in Potsdam in 2010. In April 2014, Flink was donated to the Apache Software Foundation as an incubating project. In December 2014, Flink graduated to become a top-level project of the Apache Software Foundation.
- Apache Flink is an open source stream processing framework for distributed, high-performance stream processing applications. Flink supports real-time computing with both high throughput and exactly-once semantics, and provides batch data processing.
- Unlike other data processing engines on the market, Flink and Spark support both stream and batch processing. Technically, Spark simulates stream computing based on batch processing. Flink is the opposite, simulating batch processing based on stream computing.

Key Concepts of Flink

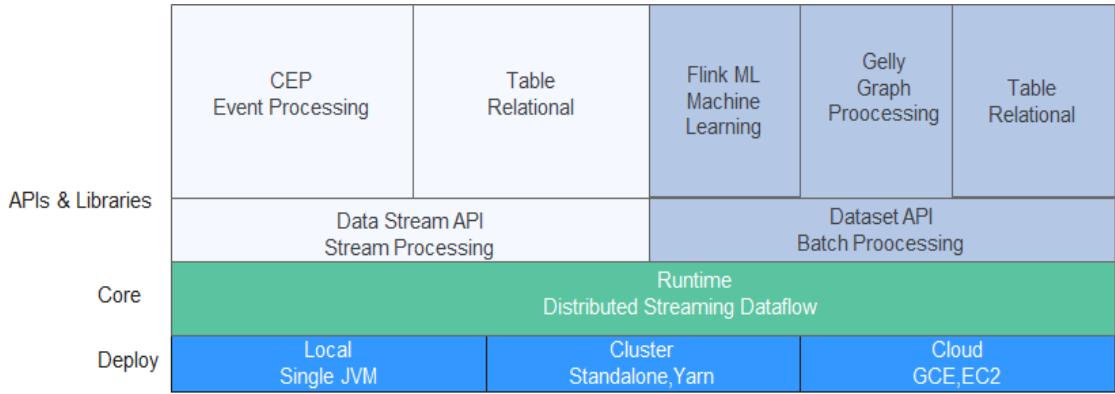
- Continuous processing of stream data
- Event time
- Stateful stream processing
- Status snapshot

Core Concepts of Flink

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

- 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.
- 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.

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.

Core Concept — 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.

- Operator operations between DataStreams
 - Window operations are associated using the **reduce**, **fold**, **sum** and **max** functions.
 - Connect: 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.

Core Concept — 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).

Flink Program

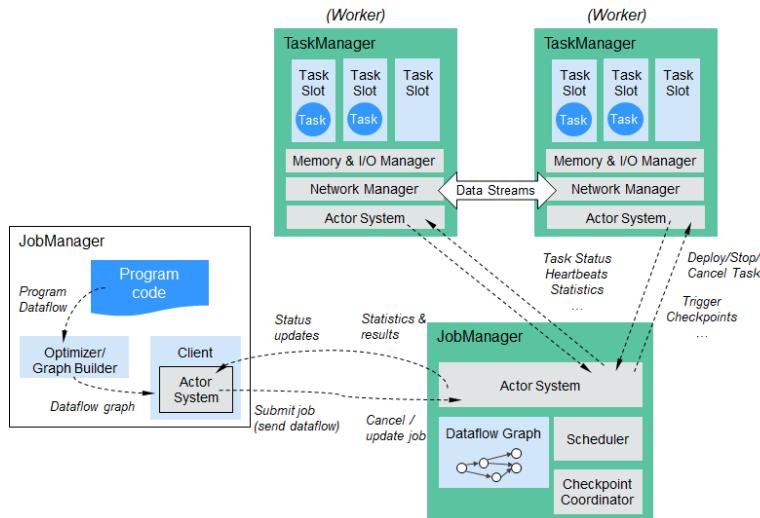
- Flink programs consist of three parts: source, transformation, and sink. The source reads data from data sources such as HDFS, Kafka, and text. The transformation is responsible for data transformation operations. The sink outputs final data (such as HDFS, Kafka, and text). Data that flows between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



Flink Data Sources

- Batch processing
 - Files
 - HDFS, local file systems, and MapR file system
 - Text, CSV, Avro, and Hadoop input formats
 - JDBC
 - HBase
 - Collections
- Stream processing
 - Files
 - Socket streams
 - Kafka
 - RabbitMQ
 - Flume
 - Collections
 - Implement your own
 - `SourceFunction.collect`

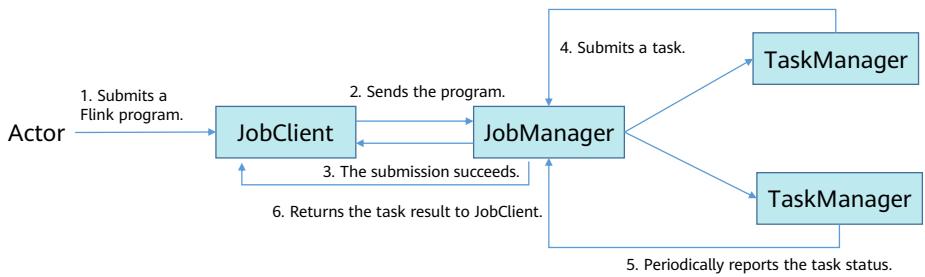
Flink Program Running



- 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 computing 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.

Flink Job Process

- 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.



Flink Job Process

- JobClient is a bridge between Flink programs and JobManager. It receives programs, parses and optimizes program execution plans, and submits them to JobManager. There are three types of operators in Flink.
 - Source Operator: data source operations, such as file, socket, and Kafka.
 - Transformation Operator: data transformation operations, such as map, flatMap, and reduce operators.
 - Sink Operator: data storage operations, for example, data is stored in HDFS, MySQL, and Kafka.

- Client: The Flink client is used to submit jobs (streaming jobs) to Flink.
- TaskManager: TaskManager is a service execution node of Flink that 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. 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.

Complete Flink Program

```
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();  
    }  
}
```

Complete Flink Program

```
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();  
    };  
};
```

- This example takes a stream of records about people as input and filters them to include adults only. In this example, **env** indicates the execution environment. Each Flink application requires an execution environment. Stream applications should use StreamExecutionEnvironment. A DataStream API call made in an application generates a job graph, and the graph is attached to StreamExecutionEnvironment. When env.execute() is called this graph packaging, it is then sent to the task manager so that it can work in parallel and be distributed to the task manager for execution. Each parallel slice of the job will be executed in the task slot.

Data Processing

- Apache Flink supports both batch and stream processing that can be used to create a number of event-based applications.
 - Flink is first of all a pure stream computing engine with data streams as its basic data model. A stream can be an unbounded infinite stream, that is, stream processing in a general sense. It can also be a bounded finite stream, that is, batch processing. Flink thus uses a single architecture to support both stream and batch processing.
 - Flink has the additional capability of supporting stateful computation. Processing is called stateless when the result of processing an event or a piece of data is only related to the content of that event itself. Alternatively, if the result is related to previously processed events, it is called stateful processing.

- Stateless computing: Stateless computing observes each independent event and generates results at the end. For example, some alarms and monitoring are used to observe each event. When the event that triggers an alarm arrives, a warning is triggered.
- Stateful computing: Stateful computing outputs results based on multiple events, such as calculating the average temperature over the past hour.
- Apache Flink is good at processing unbounded and bounded data sets. Accurate time control and stateization enable any applications that process unbounded streams to run during the Flink runtime. Bounded streams are internally processed by algorithms and data structures specifically designed for fixed-size data sets, presenting excellent performance.

Bounded Stream and Unbounded Stream

- Unbounded stream: Only the start of a stream is defined. Data sources generate data endlessly. The data of unbounded streams must be processed continuously. That is, the data must be processed immediately after being read. You cannot wait until all data arrives to process it, because the input is infinite and cannot be completed at any time. 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.
- Bounded stream: Both the start and end of a stream are defined. Bounded streams can be computed after all data is read. 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.

- There are many technologies for batch processing, including SQL processing of various relational databases and MapReduce, Hive, and Spark in the big data field. These are classic ways to handle bounded data streams. Flink focuses on unbounded stream processing. How does Flink implement batch processing? Here, we need to understand bounded and unbounded streams.
- Infinite stream processing: There is no end to input data. Data processing starts from the current or a certain time point in the past and is continuously performed.
- Finite stream processing, that is, data processing starts at one point in time and ends at another point in time. The input data may be limited (that is, the input data set does not increase over time), or may be manually set to a finite set for analysis purposes (that is, only events in a certain period of time are analyzed).
- Obviously, finite stream processing is a special case of infinite stream processing, which simply stops at a certain point in time. In addition, if the calculation results are not generated continuously during execution but only once at the end, then it is referred to as batch processing (data is processed in batches).

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

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

Example Batch Processing

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

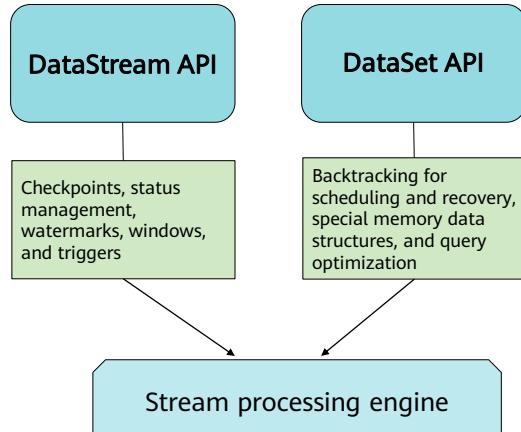
```
val counts = visits
    .keyBy("region")
    .window(GlobalWindows.create)
    .trigger(EndOfTimeTrigger.create)
    .sum("visits")
```

- 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")
```

Flink Batch Processing Model

- Flink uses an underlying engine to support both stream processing and batch processing.



Stream and Batch Processing Mechanisms

- 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.

- Table APIs and SQL use Apache Calcite to parse, verify, and optimize queries. They can be seamlessly integrated with DataStream and DataSet APIs and support user-defined scalar, aggregate, and table-valued functions.
- Data-driven applications: Transactional applications are program codes based on the experience of service personnel. Data-driven applications use computer technologies to discover rules and guide services based on accumulated service data, which is the effective utilization of enterprise data resources. Important factors of data-driven applications in practice include technology, business, data, and personnel.
- A data pipeline can be used to represent a process of implementing data migration between systems.

Contents

1. Spark — In-memory Distributed Computing Engine
2. **Flink — Stream and Batch Processing in a Single Engine**
 - Flink Principles and Architecture
 - **Flink Time and Window**
 - Flink Watermark
 - Flink Fault Tolerance Mechanism

Time Background

- In stream processor programming, time processing is very 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.

Time Classification in Stream Processing

- During data stream processing, the system time, that is, processing time, is often used as the time of an event. In fact, the processing time is the time imposed on the event. Due to reasons such as network latency, the processing time cannot reflect the sequence of events.
- In actual scenarios, the time of each event can be classified into three types:
 - Event time: the time when an event occurs.
 - Ingestion time: the time when an event arrives at the stream processing system.
 - Processing time: the time when an event is processed by the system.

- Flink supports different time concepts. Based on the location where time is generated, time is classified into processing time, event time, and ingestion time.
 - Processing time: time when an event is processed by the system. When a streaming program runs at the processing time, all time-based operations (such as the time window) use the system time of the computer that performs the operations.
 - Event time: time when an event occurs. The time is usually embedded in the event before the event enters Flink. In addition, the event timestamp can be extracted from the record of each event.
 - Ingestion time: time when an event arrives at the stream processing system. In the source operation, each record obtains the current time of the source as the timestamp. Subsequent time-based operations (such as the time window) depend on this timestamp.

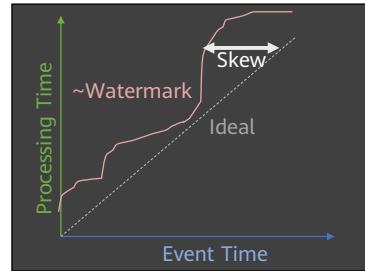
Example Time Classifications

- For example, if a log enters Flink at 10:00:00.123 on November 12, 2019 and reaches Windows at 10:00:01.234 on November 12, 2019, the log content is as follows:
 - 2019-11-02 18:37:15.624 INFO Fail over to rm2
 - 2019-11-02 18:37:15.624 is the event time.
 - 2019-11-12 10:00:00.123 is the ingestion time.
 - 2019-11-12 10:00:01.234 is the processing time.

Differences Among Three Time Classifications

- 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.

- 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.



- In Flink stream processing, most services use event time. Processing time or ingestion time is used only when event time is unavailable.

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)

- 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 is (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.

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.
- Windows are at the heart of processing infinite streams. Windows split the stream into "buckets" of finite sizes, over which we can apply computations.

Window Types

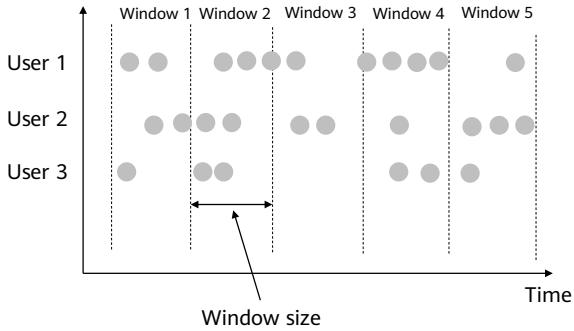
- Windows can be classified into the following types:
 - Count window: A data-driven window is generated based on the specified number of data records, which is irrelevant to time.
 - Time window: A time-driven window is generated based on time.
- Apache Flink is a distributed computing framework that supports infinite stream processing. In Flink, windows can divide infinite streams into finite streams. Flink windows can be either Time windows or Count windows.

Time Window Types

- According to the window implementation principles, time windows can be classified into:
 - Tumbling window
 - Sliding window
 - Session window

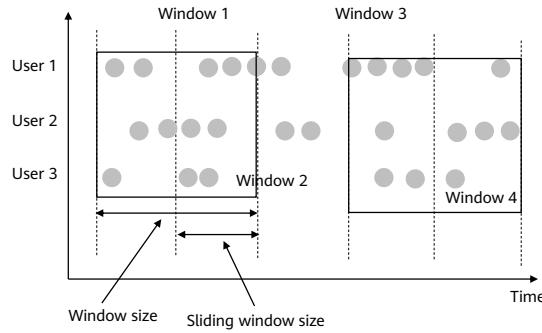
Tumbling Window

- The data is sliced according to a fixed window length.
- Characteristics: The time is aligned, the window length is fixed, and the windows do not overlap.
- Use case: BI statistics collection (aggregate calculation in each time period)



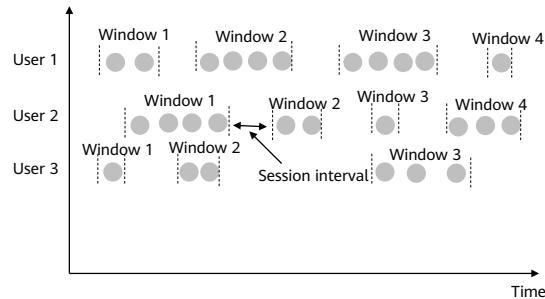
Sliding Window

- The sliding window is a more generalized form of a fixed window. It consists of a fixed window length and a sliding interval.
- Characteristics: The time is aligned, the window length is fixed, and the windows can be overlapping.
- Use case: Collect statistics on the failure rate of an API in the last 5 minutes to determine whether to report an alarm.



Session Window

- A session window consists of a series of events and a timeout interval of a specified duration. It is similar to a web application session. That is, a new window is generated if no new data is received within a period of time. Characteristics: The time is not aligned.



- The session window allocator groups elements through session activities. Compared with the tumbling window and sliding window, the session window does not have overlapping start and end times or fixed start or end time. On the contrary, **when a session window does not receive elements in a fixed time period, that is, an inactive interval is generated, then the window will be closed**. A session window is configured through a session interval. The session interval defines the length of an inactive period. When an inactive period is generated, the current session is closed and subsequent elements are allocated to a new session window.

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))`

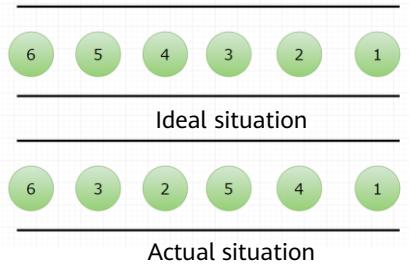
- Assume that a user performs a series of operations (such as clicking, browsing, searching, purchasing, and switching tabs) after clicking Taobao on the mobile phone. These operations and the corresponding occurrence time are sent to the server for user behavior analysis. A user behavior is a segment, and behaviors in each segment are continuous. A correlation degree of behaviors in a segment is far greater than a correlation degree of behaviors between segments. Each segment of user behavior is called a session, and the gap between segments is called a session gap. Therefore, we should, of course, divide the user's behaviors according to the session window and calculate the result of each session.
- 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.

Contents

1. Spark — In-memory Distributed Computing Engine
2. **Flink — Stream and Batch Processing in a Single Engine**
 - Flink Principles and Architecture
 - Flink Time and Window
 - **Flink Watermark**
 - Flink Fault Tolerance Mechanism

Out-of-Order Data Problem

- There is a process from event generation to stream processing through the source and then to the operator. In most cases, data is sent to the operator based on the time when the event is generated. However, out-of-order data may be generated, for example, events received by Flink are not sorted strictly based on the event time due to network problems.



Out-of-Order Data Example

- For example, an application records all clicks of a user and sends logs back. (When the network is poor, the logs are saved locally and sent back later.) User A performs operations on the application at 11:02:00, and user B performs operations on the application at 11:03:00. However, the network of user A is unstable and the log sending is delayed. As a result, the server receives the message from user B at 11:03:00 and then receives the message from user A at 11:02:00, the message is out of order.

Why Are Watermarks Needed?

- For infinite datasets, there is no effective way to determine data integrity. Therefore, watermark is a concept based on event time to describe the integrity of data streams. If events are measured by processing time, everything is ordered and perfect. Therefore, watermarks are not required. In other words, event time causes disorder. Watermarks are used to solve the disorder problem. The so-called disorder is actually an event delay. For a delayed element, it is impossible to wait indefinitely. A mechanism must be provided to ensure that the window is triggered for calculation after a specific point in time. This special mechanism is watermark, which tells operators that delayed messages should no longer be received.

- 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.

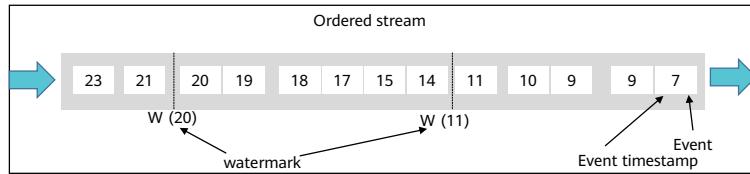
Watermark Principles

- How does Flink ensure that all data has been processed when the event time-based window is destroyed? This is what a watermark does. A watermark carries a monotonically increasing timestamp t . **Watermark(t)** indicates that all data whose timestamp is less than or equal to t has arrived, and data whose timestamp is less than or equal to t will not be received in the future. Therefore, the window can be safely triggered and destroyed.

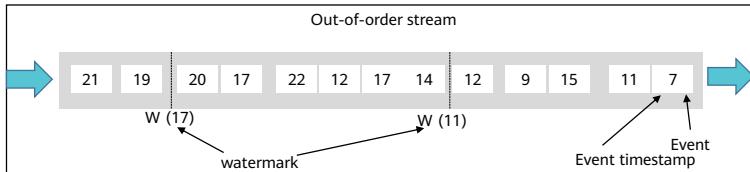
- The input data is divided to different windows based on the event time. If a window contains data and the watermark time is greater than or equal to the window maxTimestamp, the window triggering condition is met. The window triggering is determined by the window maxTimestamp in the window to which the event time of the data belongs.

Watermark Principles

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



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



Delayed Data

- Watermark can cope with out-of-order data, but a perfect watermark value cannot be obtained in the real world. Perfect values are either unobtainable or too expensive to obtain. Therefore, there is a low probability that the data before the timestamp t is received after Watermark(t). In Flink, the data is defined as late elements. You can also specify the maximum delay time in the window. The default value is 0. You can use the following code to set the maximum delay time:

```
input .keyBy(<key selector>)
    .window(<window assigner>)
    .allowedLateness(<time>)
    .<windowed transformation>(<>window function>);
```

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 employs the third method.

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, delayed data can also trigger the window for output. We can obtain this late data using the Side Output mechanism of Flink. The method is as follows:

```
final OutputTag<T> lateOutputTag = new OutputTag<T>("late-date"){};  
  
DataStream input =...;  
SingleOutputStreamOperator<T> result = input  
    .keyBy(<key selector>)  
    .window(<window assigner>)  
    .allowedLateness(<time>)  
    .<windowed transformation>(<window function>);  
DataStream<T> lateStream = result.getSideOutput(lateOutputTag);
```

- 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 the delayed data and perform special processing. After Allowed Lateness is set, delayed data can also trigger the window for output. Using the Side Output mechanism of Flink, users can obtain the delayed data.
- Note that after **allowedLateness** is, delayed data may trigger windows. For a session window, windows may be merged, resulting in unexpected behavior.
- **Window**: window type; **allowedLateness**: allowed delay time.

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.

Contents

1. Spark — In-memory Distributed Computing Engine
2. **Flink — Stream and Batch Processing in a Single Engine**
 - Flink Principles and Architecture
 - Flink Time and Window
 - Flink Watermark
 - **Flink Fault Tolerance Mechanism**

Checkpointing

- How does Flink ensure exactly-once semantics?
 - Flink uses a feature called checkpoint to reset the system to the correct state when a fault occurs.
 - Flink offers the fault tolerance mechanism based on the lightweight asynchronous snapshot technology for distributed data streams. It can globally take snapshots of the state data of tasks or operations at the same time point, that is, backs up the states of programs.

- If a program is faulty, you need to restore the state data. The state fault tolerance can be implemented only when the program provides support. Flink provides a checkpoint fault tolerance mechanism to ensure exactly-once semantics.
- Flink stores the state data of a specific task or operation in the Java heap memory by default.
- A checkpoint is a global state snapshot of a Flink job at a specific time point and contains the states of all tasks or operations. The checkpoint stores the state data persistently.

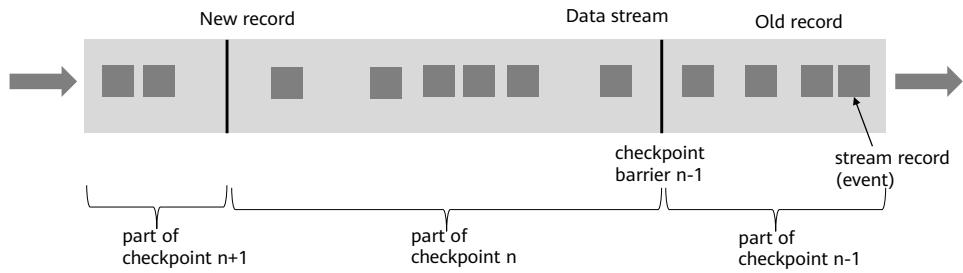
Checkpoint Mechanism

- Flink provides the checkpoint fault tolerance mechanism. The distributed snapshot function can take global snapshots of task/operator status data at the same time point in a unified manner. 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 stream processing applications, these snapshots are very lightweight and can be taken frequently without compromising performance.
 - 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).

- For example, the offset status is maintained in the KafkaConsumer operator. When the system is faulty and cannot consume data from Kafka, the offset can be recorded in the status. When the task is restored, the consumption can start from the specified offset.

Checkpointing Mechanism

- A core element in Flink's distributed snapshotting is stream barriers. These barriers are injected into the data stream and flow with the records as part of the data stream. Barriers flow strictly in line. A barrier separates the records in the data stream into the set of records that goes into the current snapshot, and the records that go into the next snapshot. Barriers do not interrupt the flow of the stream and are hence very lightweight. Multiple barriers from different snapshots can be in the stream at the same time, which means that various snapshots may happen concurrently.



Checkpointing Configuration

- 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.
```

Checkpointing Configuration

- Exactly-once or at-least-once
 - Exactly-once ensures end-to-end data consistency, prevents data loss and duplicates, but 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 Configuration

- 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)
```

Checkpointing Configuration

- Minimum interval between checkpoints
 - When checkpoints frequently end up taking longer than the base interval because the state grew larger than planned, the system constantly takes checkpoints. This can mean that too many compute 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)
```

Checkpointing Configuration

- 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(500)
```

Checkpointing Configuration

- 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(ExternalizedCheckpointCleanup.RETAIN_ON_CANCELLATION)
```

How Do I Restore Data for a Job?

- 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.

Savepoint vs. Checkpoint

Parameter	Checkpoint	Savepoint
Trigger and Management Method	Automatically triggered and managed by Flink	Manually triggered and managed by users
Usage	Quickly recovers tasks when exceptions occur, for example, timeout exceptions caused by network jitter.	Backs up data as planned, for example, modifying code and adjusting concurrency.
Characteristics	<ul style="list-style-type: none">LightweightAutomatic restoration from faultsCleared by default after a job is stopped	<ul style="list-style-type: none">PersistentStored in a standard format, allowing code or configuration changesManually triggered for restoration from a savepoint

- 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.

State Backend

- Programs written in the DataStream API often hold their states in various forms. When checkpointing is activated, such states are persisted upon checkpoints to prevent against data loss and ensure data consistency in recovery. How a state is represented internally, and how and where it is persisted upon checkpoints depends on the chosen state backend.

State Storage Method — MemoryStateBackend

- Construction method
 - `MemoryStateBackend(int maxStateSize, boolean asynchronousSnapshots)`
- Storage method
 - State: TaskManager memory
 - Checkpoint: JobManager memory
- Capacity limit
 - The default value of `maxStateSize` for a single state is 5 MB.
 - `maxStateSize ≤ akka.framesize` (Default value: 10 MB)
 - The total size cannot exceed the memory of JobManager.
- The `MemoryStateBackend` is encouraged for local testing and jobs that hold few states, such as ETL.

- The first state storage method of checkpoints is memory storage, that is, `MemoryStateBackend`. The construction method is to configure `maxStateSize` and determine whether to perform asynchronous snapshot operations. Storage states are stored in the memory of the TaskManager node, that is, the memory of the execution node. Because the memory capacity is limited, the default value of `maxStateSize` for a single state is 5 MB, and the value of `maxStateSize` must be less than or equal to the value of `akka.framesize` (10 MB by default). Since the JobManager memory stores checkpoints, the checkpoint size cannot be larger than the memory of the 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. The `MemoryStateBackend` is not recommended in production scenarios.

State Storage Method — FsStateBackend

- Construction method
 - `FsStateBackend(URL checkpointDataUri, boolean asynchronousSnapshots)`
- Storage method
 - State: TaskManager memory
 - Checkpoint: external file storage system (local or HDFS)
- Capacity limit
 - The size of states on a single TaskManager cannot exceed its memory.
 - The total size cannot exceed the configured file system capacity.
- The FsStateBackend is recommended for jobs with common states, such as aggregation at the minute-window level and join, and jobs requiring high-availability setups. It can be used in production scenarios.

- Another state storage method is FsStateBackend on the file system. The construction method is to transfer a file path and whether to perform asynchronous snapshot operations. The FsStateBackend also holds state data in the memory of the TaskManager, but unlike MemoryStateBackend, it does not 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.

State Storage Method — RocksDBStateBackend

- Construction method
 - RocksDBStateBackend(URI checkpointDataUri, boolean enableIncrementalCheckpointing)
- Storage mode
 - State: KV database on the TaskManager (used memory + disk)
 - Checkpoint: external file storage system (local or HDFS)
- Capacity limit
 - The size of states on a single TaskManager cannot exceed the memory and disk size of the TaskManager.
 - The maximum size of a key is 2 GB.
- The RocksDBStateBackend is recommended for jobs with very large states, for example, aggregation at the day-window level, jobs requiring high-availability setups, and jobs that do not require high read/write performance. It can be used in production scenarios.

- Another storage method is 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.

Summary

- The first part of this chapter describes the basic concepts and technical architecture of Spark, including SparkSQL, Structured Streaming, and Spark Streaming. The second part describes the architecture and technical principles of Flink and the running process of Flink programs. The focus is on the difference between Flink stream processing and batch processing. In the long run, the DataStream API should completely include the DataSet API through bounded data streams.

Quiz

1. (Multiple-choice) Which of the following are Flink components? ()
 - A. JobManager
 - B. TaskManager
 - C. ResourceManager
 - D. Dispatcher
2. (Single-choice) Flink stores the state data of a specific task or operation in HDFS by default. ()
 - A. True
 - B. False

- Answer: 1. ABCD 2. B

Quiz

3. (Single-choice) The sliding window is a more generalized form of a fixed window. It consists of a fixed window length and a sliding interval. ()
 - A. True
 - B. False
4. (Multiple-choice) Which of the following are state storage methods provided by Flink? ()
 - A. MemoryStateBackend
 - B. FsStateBackend
 - C. RocksDBStateBackend
 - D. HDFS
5. (Single-choice) Flink provides three layered APIs. Each API offers a different trade-off between conciseness and expressiveness and targets different use cases. ()
 - A. True
 - B. False

- Answer: 3. A 4. ABC 5. A

Acronyms and Abbreviations

- ETL: extract, transform, and load, a process that involves extracting data from sources, transforming the data, and loading the data to final targets
- FIFO: First Input First Output
- SQL: Structured Query Language
- Write-Ahead Logging (WAL)
- BI: Business Intelligence
- API: Application Programming Interface
- APP: Application
- SQL: Structured Query Language

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive
statements including, without limitation, statements regarding
the future financial and operating results, future product
portfolio, new technology, etc. There are a number of factors
that could cause actual results and developments to differ
materially from those expressed or implied in the predictive
statements. Therefore, such information is provided for reference
purpose only and constitutes neither an offer nor an acceptance.
Huawei may change the information at any time without notice.



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



Foreword

- This chapter consists of the following two parts:
 - The first part is about Flume. It is a distributed, reliable, and highly available massive log aggregation system that supports custom data transmitters for collecting data. It also roughly processes data and writes the data to customizable data receivers.
 - The second part describes the basic concepts, architecture, and functions of Kafka. It is important to know how Kafka ensures reliable data storage and transmission and how historical data is processed.

Objectives

- Upon completion of this chapter, you will be able to:
 - Know what is Flume.
 - Understand the system architecture of Flume.
 - Grasp key features of Flume.
 - Master Flume applications.
 - Be familiar with the basic concepts of Kafka.
 - Understand the system architecture of Kafka.

Contents

1. Flume: Massive Log Aggregation

- Overview and Architecture
- Key Features
- Applications

2. Kafka: Distributed Messaging System

What Is Flume?

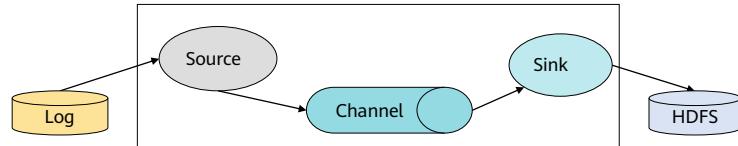
- Flume is a streaming log collection tool that roughly processes data and writes the data to data receivers. It collects data from local files (spooling directory source), real-time logs (**taildir** and **exec**), REST messages, Thrift, Avro, Syslog, Kafka, and other data sources.

What Can Flume Do?

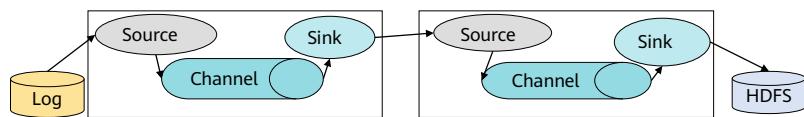
- Collects log information from a fixed directory to a destination (such as HDFS, HBase, or Kafka).
- Collects log information (such as **taildir**) to a destination in real time.
- Supports cascading (connecting multiple Flumes) and data conflation.
- Supports custom data collection tasks based on users.

Flume Architecture

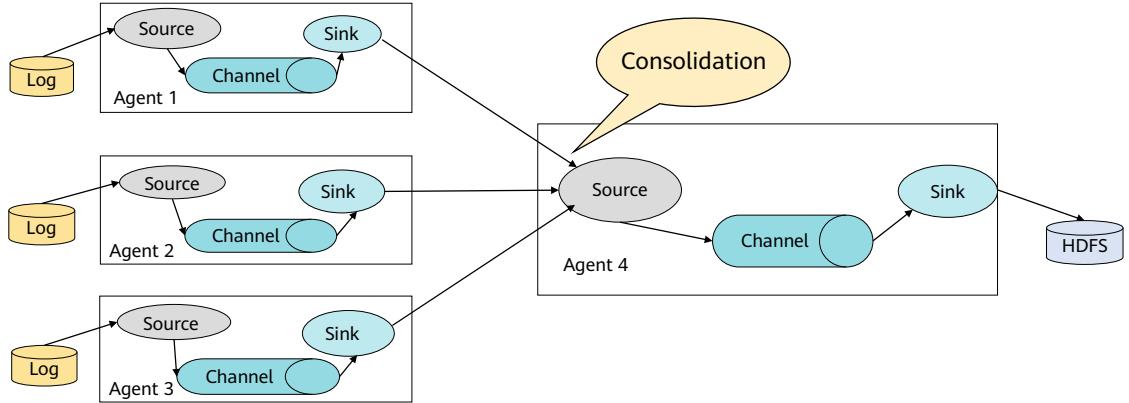
- Infrastructure: Flume can directly collect data with an agent, which is mainly for data collection in a cluster.



- Multi-agent architecture: Flume can connect multiple agents to collect raw data and store it in the final storage system. This architecture is used to import data into the cluster.

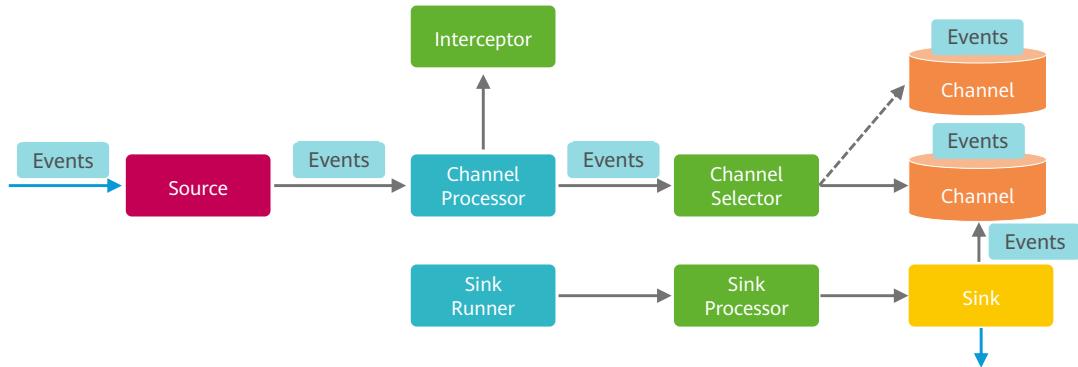


Flume Multi-Agent Consolidation



- You can configure multiple level-1 agents and point them to the source of a level-2 agent using Flume. The source of the level-2 agent consolidates the received events and sends the consolidated events into a single channel. The events in the channel are consumed by a sink and then pushed to the destination.

Flume Architecture Diagram



Basic Concepts — Source (1)

- A source receives or generates events using a special mechanism and places them to one or more channels in batches.
- There are two types of sources: driver-based and polling.
 - Driver-based sources: External systems proactively send data to Flume, driving Flume to receive data.
 - Polling sources: Flume periodically obtains data.
- A source must be associated with at least one channel.

- Source: data source, that is, the source of log information. Flume models the raw data and abstracts it into the processed data objects (events).

Basic Concepts — Source (2)

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.
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.

Basic Concepts — Channel (1)

- Channels are located between a source and sink. They function as queues and are used to cache temporary events. When a sink successfully sends events to the channel of the next hop or the final destination, the events are removed from the channel.
- Channels are fully transactional, provide weak ordering guarantees, and can work with any number of sources and sinks.
- Different channels offer different levels of persistence:
 - Memory channel: volatile
 - File channel: backed by write-ahead log (WAL) implementation
 - JDBC channel: implemented based on embedded databases

- Channels support transactions. If an exception occurs when the channel data (usually batch data) on Flume is sent to the next phase, the data is rolled back. The data is still stored in the channel and waits for the next reprocessing.

Basic Concepts — Channel (2)

- 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 (for each file channel) and checkpoint directory.
- 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.

Basic Concepts — Sink

- A sink sends events to the next hop or the final destination, and then removes the events from the channel.
- A sink must work with a specific channel.

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.
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.

Contents

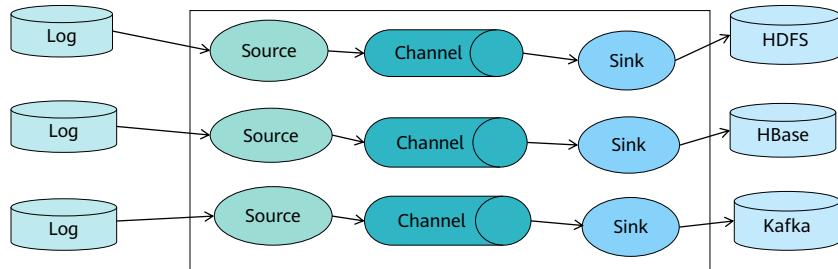
1. Flume: Massive Log Aggregation

- Overview and Architecture
- Key Features
- Applications

2. Kafka: Distributed Messaging System

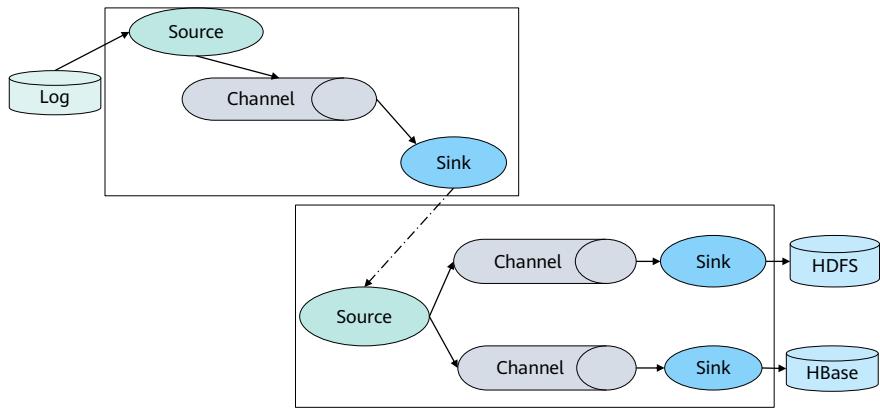
Log File Collection

- Flume collects log files in a cluster and archives them in HDFS, HBase, or Kafka used for data analysis and cleansing for upper-layer applications.



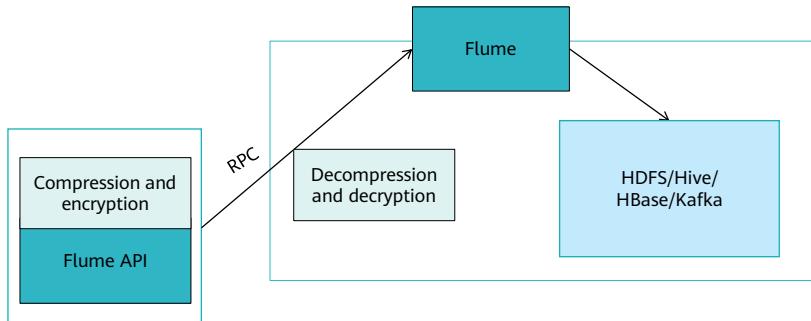
Multi-level Cascading and Multi-channel Replication

- Flume supports cascading of multiple Flume agents and data replication within the cascaded agents.



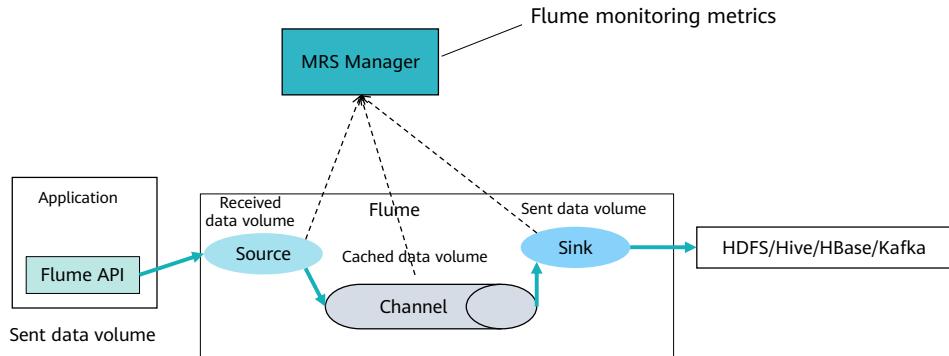
Cascading Message Compression and Encryption

- Data transmission between cascaded Flume agents can be compressed and encrypted, improving data transmission efficiency and security.



- In a non-cascading architecture, when data is sent from a sink to a storage medium, whether encryption is required depends on the storage medium and sink type.
- 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.

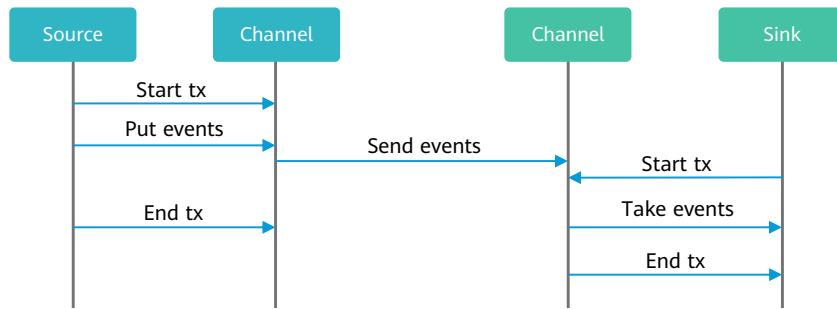
Data Monitoring



- MRS Manager visually displays monitoring metrics such as the received data volume of the source, cached data volume in the channel, and data volume written into the sink.
- Flume offers channel-based data caching, data sending, and data receiving failure alarms.

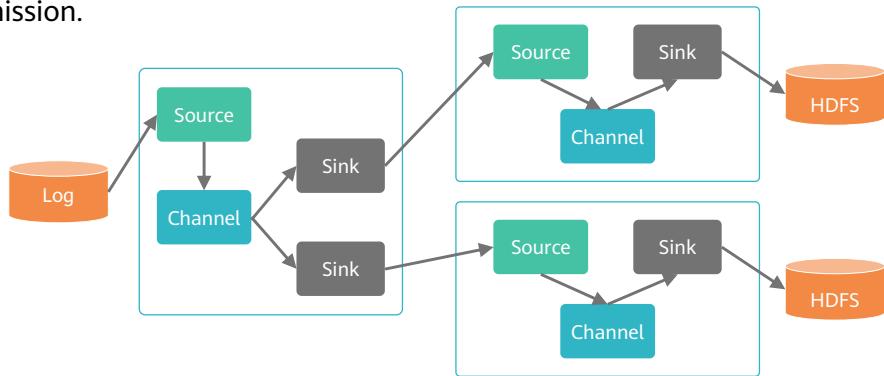
Transmission Reliability

- During data transmission, 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.



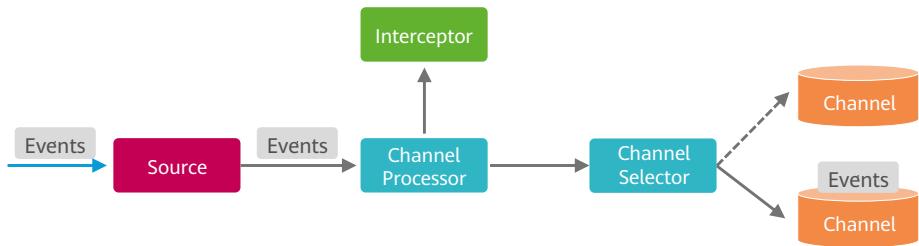
Failover

- 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.



Data Filtering During Transmission

- During data transmission, Flume roughly filters and cleans data and deletes unnecessary data. If the data to be filtered is complex, users need to develop filter plugins based on their data characteristics. Flume supports third-party filter plugins.



Contents

1. Flume: Massive Log Aggregation

- Overview and Architecture
- Key Features
- Applications

2. Kafka: Distributed Messaging System

Flume Operation Example 1 (1)

- Description
 - This example shows how Flume ingests logs generated by applications (such as e-banking systems) in a cluster to HDFS.
- Prepare data.
 - Create a log directory named **/tmp/log_test** on a node in the cluster.
 - Use this directory as the monitoring directory.
- Install the Flume client.
 - Log in to MRS Manager and download the Flume client.
 - Decompress the package.
 - Install the client.

- In this example, logs in a directory are collected and archived to HDFS.

Flume Operation Example 1 (2)

- Configure the Flume source.

```
server.sources = a1
server.channels = ch1
server.sinks = s1
# the source configuration of a1
server.sources.a1.type = spooldir
server.sources.a1.spoolDir = /tmp/log_test
server.sources.a1.fileSuffix = .COMPLETED
server.sources.a1.deletePolicy = never
server.sources.a1.trackerDir = .flumespool
server.sources.a1.ignorePattern = ^$ 
server.sources.a1.batchSize = 1000
server.sources.a1.inputCharset = UTF-8
server.sources.a1.deserializer = LINE
server.sources.a1.selector.type = replicating
server.sources.a1.fileHeaderKey = file
server.sources.a1.fileHeader = false
server.sources.a1.channels = ch1
```

- server.sources.a1.fileSuffix** indicates the suffix added to the file after the ingestion is completed. (The suffix indicates that the file has been ingested and will not be ingested next time.)
- server.sources.a1.deletePolicy** indicates the policy for deleting source files after file transfer. The value can be **never** or **immediate**. By default, source files are not deleted.
- server.sources.a1.trackerDir** indicates the path for storing the metadata of files ingested by Flume.
- server.sources.a1.ignorePattern** indicates the regular expression for the names of files that do not need to be ingested in the ingestion path.
- server.sources.a1.batchSize** indicates the number of events that Flume sends in a batch (the number of data records). A larger value indicates higher performance and lower timeliness.
- server.sources.a1.deserializer** indicates the file reading mode of Flume.
- server.sources.a1.selector.type** indicates the mode (replicating or multiplexing) for Flume to send data to channels.
- server.sources.a1.fileHeaderKey** indicates the key name of the absolute path of the file where the current event is stored in the event (data) header.
- server.sources.a1.fileHeader** indicates whether to store the file absolute path in the event (data) header. The value can be **true** or **false**.

Flume Operation Example 1 (3)

- Configure the Flume channel.

```
# the channel configuration of ch1
server.channels.ch1.type = memory
server.channels.ch1.capacity = 10000
server.channels.ch1.transactionCapacity = 1000
server.channels.ch1.channlefullcount = 10
server.channels.ch1.keep-alive = 3
server.channels.ch1.byteCapacityBufferPercentage = 20
```

- server.channels.ch1.type** indicates the channel type. The value can be **file** or **memory**.
- server.channels.ch1.capacity** indicates the number of events cached in the buffer. The value cannot be too large. It is recommended that the value be 10 times of the value of **transactionCapacity**.
- server.channels.ch1.transactionCapacity** indicates the transaction size, that is, the number of events in a transaction that can be processed by the current channel. The size cannot be smaller than the **batchSize** of the source data. You are advised to set the same size as **batchSize**.
- server.channels.ch1.channlefullcount** indicates the channel full count. When the value reaches the threshold, an alarm is reported.
- server.channels.ch1.keep-alive** indicates the timeout interval for adding or deleting an event in the buffer.
- server.channels.ch1.byteCapacityBufferPercentage** indicates the percentage of the remaining buffer memory. When the remaining buffer memory is less than the value of **byteCapacity**, data cannot be sent to the buffer.

Flume Operation Example 1 (4)

- Configure the Flume sink.

```
server.sinks.s1.type = hdfs
server.sinks.s1.hdfs.path = /tmp/flume_avro
server.sinks.s1.hdfs.filePrefix = over_%{basename}
server.sinks.s1.hdfs.inUseSuffix = .tmp
server.sinks.s1.hdfs.rollInterval = 30
server.sinks.s1.hdfs.rollSize = 1024
server.sinks.s1.hdfs.rollCount = 10
server.sinks.s1.hdfs.batchSize = 1000
server.sinks.s1.hdfs.fileType = DataStream
server.sinks.s1.hdfs.maxOpenFiles = 5000
server.sinks.s1.hdfs.writeFormat = Writable
server.sinks.s1.hdfs.callTimeout = 10000
server.sinks.s1.hdfs.threadsPoolSize = 10
server.sinks.s1.hdfs.failcount = 10
server.sinks.s1.hdfs.fileCloseByEndEvent = true
server.sinks.s1.channel = ch1
```

- server.sinks.s1.type** indicates the sink type. The value can be **hdfs**, **hbase**, **kafka**, **avro**, or **solr**.
- server.sinks.s1.hdfs.path** indicates the HDFS data write directory. This parameter value cannot be left blank.
- server.sinks.s1.hdfs.filePrefix** indicates the prefix of a file after it is written to HDFS.
- server.sinks.s1.hdfs.inUseSuffix** indicates the prefix of a file that is being written to HDFS.
- server.sinks.s1.hdfs.rollInterval** indicates the interval for creating a file in HDFS.
- server.sinks.s1.hdfs.rollSize** indicates the size (unit: byte) of a file written to HDFS before a new file is created.
- server.sinks.s1.hdfs.rollCount** indicates the number of events written to HDFS before a new file is created.
- server.sinks.s1.hdfs.batchSize** indicates the maximum number of events that can be written to HDFS once.
- server.sinks.s1.hdfs.fileType** indicates the file format (serialized file, byte file, and compressed file) written to HDFS.
- server.sinks.s1.hdfs.maxOpenFiles** indicates the maximum number of files that can be opened in HDFS.
- server.sinks.s1.hdfs.writeFormat** indicates the serialization mode of the serialized file written to HDFS. The value can be **Text** or **Writable**.
- server.sinks.s1.hdfs.callTimeout** indicates the timeout interval for operations on HDFS.
- server.sinks.s1.hdfs.threadsPoolSize** indicates the size of the thread pool for writing data to HDFS.
- server.sinks.s1.hdfs.failcount** indicates the number of consecutive failures to write data to HDFS.
- server.sinks.s1.hdfs.fileCloseByEndEvent** indicates whether to close the file that is being written to HDFS based on the source file data. The value is **true** or **false**. When configuring options related to roll, the value should be **false**. Otherwise, the configurations related to roll are invalid.
- server.sinks.s1.channel** indicates the channel to which the data read by the current sink is sent. This parameter cannot be left blank.

Flume Operation Example 1 (5)

- Name the configuration file of the Flume agent **properties.properties**, and upload the configuration file.
- Produce data in the **/tmp/log_test** directory.

```
mv /var/log/log.11 /tmp/log_test
```

- Check whether HDFS has data obtained from the sink.

```
hdfs dfs -ls /tmp/flume_avro
```

- In this case, **log.11** is renamed **log.11.COMPLETED** by Flume, indicating that the ingestion is successful.

- Run the **chmod -R 666 /tmp/log_test** command to grant the read and write permissions on the **/tmp/log_test** directory.
- You are advised to disable Flume when uploading the configuration file. After the upload, start Flume.

Flume Operation Example 2 (1)

- Description
 - This example shows how Flume ingests clickstream logs to Kafka in real time for subsequent analysis and processing.
- Prepare data.
 - Create a log directory named `/tmp/log_click` on a node in the cluster.
 - Ingest data to Kafka topic_1028.

Flume Operation Example 2 (2)

- Configure the Flume source.

```
server.sources = a1
server.channels = ch1
server.sinks = s1
# the source configuration of a1
server.sources.a1.type = spooldir
server.sources.a1.spoolDir = /tmp/log_click
server.sources.a1.fileSuffix = .COMPLETED
server.sources.a1.deletePolicy = never
server.sources.a1.trackerDir = .flumespool
server.sources.a1.ignorePattern = ^$ 
server.sources.a1.batchSize = 1000
server.sources.a1.inputCharset = UTF-8
server.sources.a1.selector.type = replicating
jserver.sources.a1.basenameHeaderKey = basename
server.sources.a1.deserializer.maxBatchLine = 1
server.sources.a1.deserializer.maxLineLength = 2048
server.sources.a1.channels = ch1
```

- server.sources.a1.fileSuffix** indicates the suffix added to the file after the ingestion is completed. (The suffix indicates that the file has been ingested and will not be ingested next time.)
- server.sources.a1.deletePolicy** indicates the policy for deleting source files after file transfer. The value can be **never** or **immediate**. By default, source files are not deleted.
- server.sources.a1.trackerDir** indicates the path for storing the metadata of files ingested by Flume.
- server.sources.a1.ignorePattern** indicates the regular expression for the names of files that do not need to be ingested in the ingestion path.
- server.sources.a1.batchSize** indicates the number of events that Flume sends in a batch (the number of data records). A larger value indicates higher performance and lower timeliness.
- server.sources.a1.deserializer** indicates the file reading mode of Flume.
- server.sources.a1.selector.type** indicates the mode (replicating or multiplexing) for Flume to send data to channels.
- server.sources.a1.fileHeaderKey** indicates the key name of the absolute path of the file where the current event is stored in the event (data) header.
- server.sources.a1.fileHeader** indicates whether to store the file absolute path in the event (data) header. The value can be **true** or **false**.

Flume Operation Example 2 (3)

- Configure the Flume channel.

```
# the channel configuration of ch1
server.channels.ch1.type = memory
server.channels.ch1.capacity = 10000
server.channels.ch1.transactionCapacity = 1000
server.channels.ch1.channlefullcount = 10
server.channels.ch1.keep-alive = 3
server.channels.ch1.byteCapacityBufferPercentage = 20
```

- server.channels.ch1.type** indicates the channel type. The value can be **file** or **memory**.
- server.channels.ch1.capacity** indicates the number of events cached in the buffer. The value cannot be too large. It is recommended that the value be 10 times of the value of **transactionCapacity**.
- server.channels.ch1.transactionCapacity** indicates the transaction size, that is, the number of events in a transaction that can be processed by the current channel. The size cannot be smaller than the **batchSize** of the source data. You are advised to set the same size as **batchSize**.
- server.channels.ch1.channlefullcount** indicates the channel full count. When the value reaches the threshold, an alarm is reported.
- server.channels.ch1.keep-alive** indicates the timeout interval for adding or deleting an event in the buffer.
- server.channels.ch1.byteCapacityBufferPercentage** indicates the percentage of the remaining buffer memory. When the remaining buffer memory is less than the value of **byteCapacity**, data cannot be sent to the buffer.

Flume Operation Example 2 (4)

- Configure the Flume sink.

```
# the sink configuration of s1
server.sinks.s1.type = org.apache.flume.sink.kafka.KafkaSink
server.sinks.s1.kafka.topic = topic_1028
server.sinks.s1.flumeBatchSize = 1000
server.sinks.s1.kafka.producer.type = sync
server.sinks.s1.kafka.bootstrap.servers = 192.168.225.15:21007
server.sinks.s1.kafka.security.protocol = SASL_PLAINTEXT
server.sinks.s1.requiredAcks = 0
server.sinks.s1.channel = ch1
```

- server.sinks.s1.kafka.topic** indicates the name of a Kafka topic. The default value is **default-flume-topic**.
- server.sinks.s1.flumeBatchSize** indicates the number of events that Flume sends in batches (the number of data records).
- server.sinks.s1.kafka.producer.type** indicates the mode in which Flume sends data. The value is **sync** (synchronous) or **async** (asynchronous).
- server.sinks.s1.kafka.bootstrap.servers** indicates the bootstrap IP address and port list of Kafka. The default value is all lists in a Kafka cluster. If Kafka has been installed in the cluster and its configurations have been synchronized, this parameter can be left blank.
- server.sinks.s1.kafka.security.protocol** indicates the Kafka security protocol. Set this parameter to **SASL_PLAINTEXT** for a security Kafka and **PLAINTEXT** for a non-security Kafka. The default value is **SASL_PLAINTEXT**. This parameter can be left blank for a security cluster. For a non-security cluster, set it to **PLAINTEXT**.
- server.sinks.s1.requiredAcks** indicates the number of acknowledgements (ACKs) that the producer waits for from Kafka.

Flume Operation Example 2 (5)

- Upload the configuration file to Flume.
- Run the Kafka command to view the data ingested from Kafka topic_1028.

```
fi02host01:/opt/hadoopclient_0810/Kafka/kafka # bin/kafka-console-consumer.sh \
--topic topic_1028 --bootstrap-server 192.168.225.15:21007 \
--new-consumer --consumer.config config/consumer.properties
{"movie":"914","rate":"3","timeStamp":"978301968","uid":"1"}
 {"movie":"594","rate":"4","timeStamp":"978302268","uid":"1"}
 {"movie":"1197","rate":"3","timeStamp":"978302268","uid":"1"}
 {"movie":"2355","rate":"5","timeStamp":"978824291","uid":"1"}
 {"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"}
 {"movie":"2804","rate":"5","timeStamp":"978300719","uid":"1"}
 {"movie":"3408","rate":"4","timeStamp":"978300275","uid":"1"}
 {"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}
 {"movie":"1287","rate":"5","timeStamp":"978302039","uid":"1"}
```

Contents

1. Flume: Massive Log Aggregation
2. **Kafka: Distributed Messaging System**
 - Overview
 - Architecture and Functions
 - Data Management

Introduction

- Kafka is a distributed, partitioned, replicated, and ZooKeeper-based messaging system. It supports multi-subscribers and was originally developed by LinkedIn.
- Main application scenarios: log collection systems and messaging systems
- 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.

- Kafka was originally developed by LinkedIn and open sourced in 2011.
- Kafka is written in Scala and Java.
- Kafka is used at LinkedIn for activity stream and operational data processing pipelines.
- Kafka has been used by many companies for various data pipelines and messaging systems.

Point-to-Point Messaging

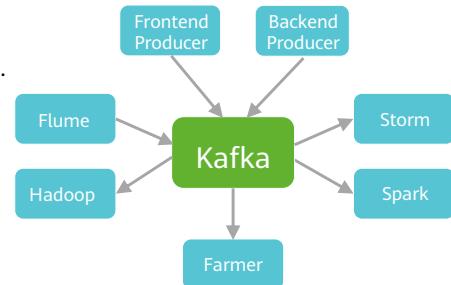
- In a point-to-point messaging system, messages are persisted in a queue. In this case, one or more consumers consume the messages in the queue. However, a message can be consumed by a maximum of one consumer only. Once a consumer reads a message in the queue, the message disappears from that queue.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

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.

Kafka Features

- Kafka can persist messages in a time complexity of $O(1)$, and can maintain data access performance in constant time even in the face of terabytes of data.
- Kafka provides high throughput. Even on cheap commercial machines, a single-node system can transmit 100,000 messages per second.
- Kafka supports message partitioning and distributed consumption, and ensures that messages are transmitted in sequence in each partition.
- Kafka supports offline and real-time data processing.
- Kafka supports online horizontal scale-out.

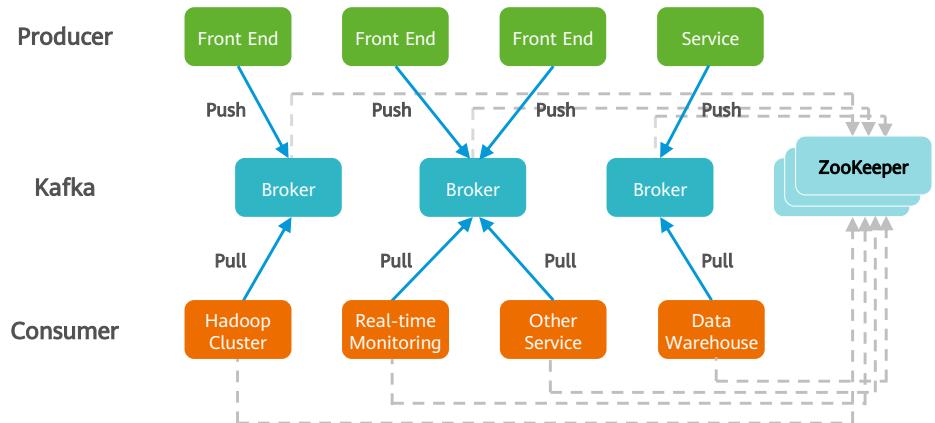


- The time complexity is $O(1)$. It is the lowest spatial-temporal complexity. That is, how much time/space is consumed does not depend on the size of the input. No matter how many times the input increases, the time/space consumption remains unchanged.

Contents

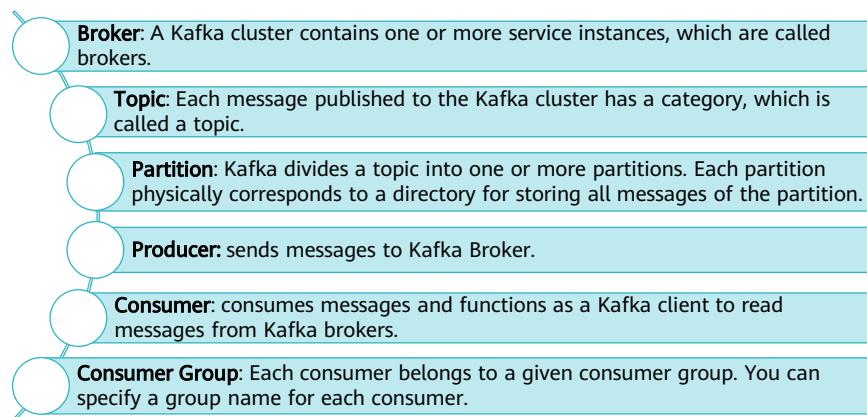
1. Flume: Massive Log Aggregation
2. **Kafka: Distributed Messaging System**
 - Overview
 - Architecture and Functions
 - Data Management

Kafka Topology



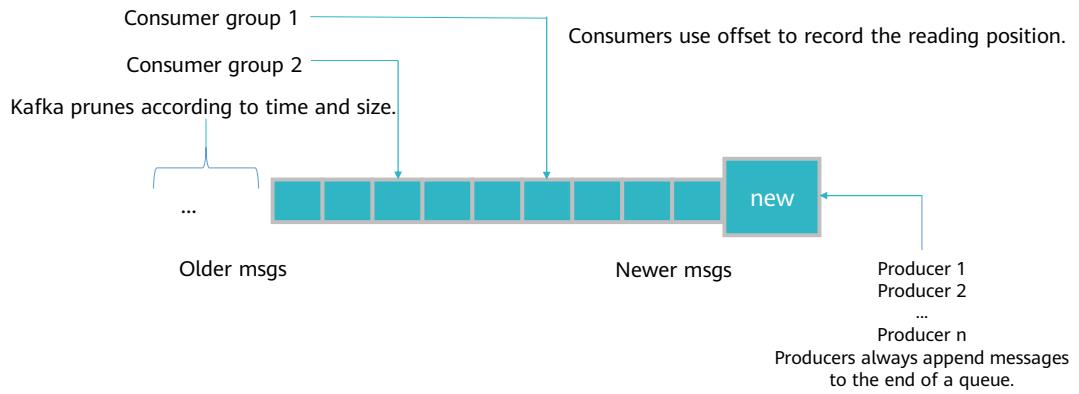
- A Kafka cluster typically consists of multiple producers (such as page views produced in the web frontend, server logs, system CPUs, and memory), brokers (Kafka supports scale-out. More brokers, higher throughput rate of the cluster), consumers, and a ZooKeeper cluster. Kafka uses ZooKeeper to manage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

Kafka Basic Concepts



Kafka Topics

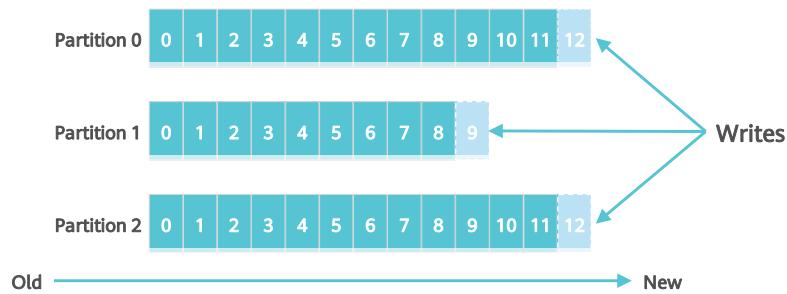
- Each message published to Kafka belongs to a category, which is called a topic. A topic can also be interpreted as a message queue. For example, weather can be regarded as a topic (queue) that stores daily temperature information.



- Each message published to Kafka belongs to a category, which is called a topic. A topic can also be interpreted as a message queue. For example, weather can be regarded as a topic (queue) that stores daily temperature information.
- In this figure, the blue box indicates a topic of Kafka, that is, a queue. Each box indicates a message. Messages generated by a producer are placed at the end of the topic one by one. A consumer reads messages from left to right. Offset is used to denote the reading position of the consumer.

Kafka Partitions

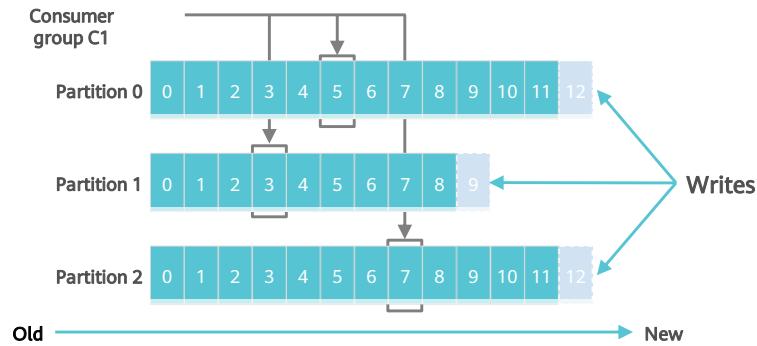
- To improve the throughput of Kafka, each topic is physically divided into one or more partitions. Each partition is an ordered and immutable sequence of messages. Each partition physically corresponds to a directory for storing all messages and index files of the partition.



- Each topic is divided into multiple partitions. Each partition corresponds to a log file at the storage layer. A log file records all messages.
- Multiple partitions of a topic are distributed on different Kafka nodes, ensuring high throughput capabilities of Kafka. Therefore, multiple clients (producers and consumers) can concurrently access different nodes to read and write messages of a topic.

Kafka 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.

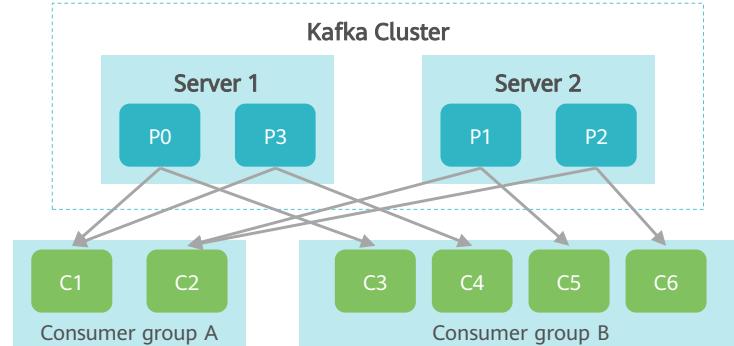


Offset Storage Mechanism

- 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.
- Offsets of consumer groups are stored in the `_consumer_offsets` directory.
 - Formula: `Math.abs(groupId.hashCode()) % 50`
 - Go to the `kafka-logs` directory and you will find multiple sub-directories. This is because kafka generates 50 `_consumer_offsets-n` directories by default.

Consumer Groups

- 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.



Other Important Concepts

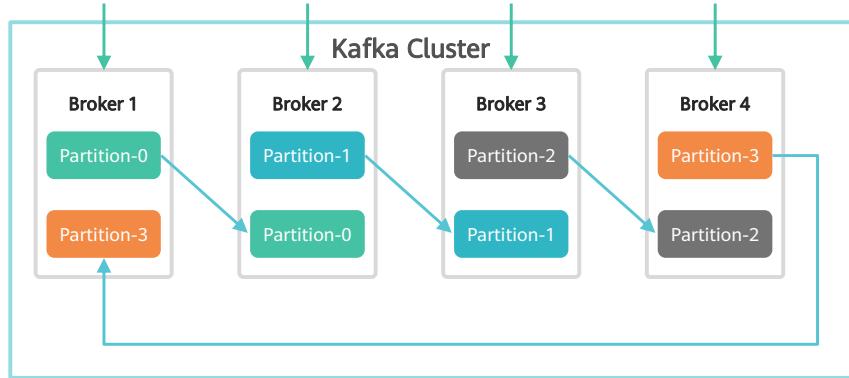
- Replica:
 - Refers to a replica of a partition, which guarantees high availability of partitions.
- Leader:
 - A role in a replica. Producers and consumers only interact with the leader.
- Follower:
 - A role in a replica, which replicates data from the leader.
- Controller:
 - A server in a Kafka cluster, which is used for leader election and failovers.

Contents

1. Flume: Massive Log Aggregation
2. **Kafka: Distributed Messaging System**
 - Overview
 - Architecture and Functions
 - Data Management

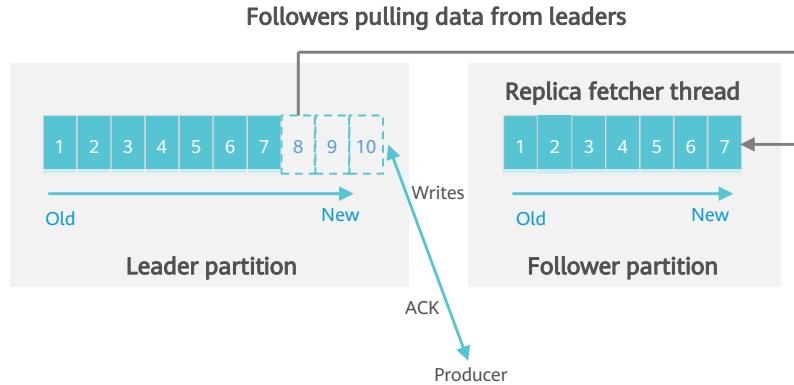
Partition Replica

- Each partition has one or more replications, which are distributed on different brokers of the cluster to improve availability. From the storage perspective, each partition replication is logically abstracted as a log object, meaning partition replications correspond to specific log objects.



Master-Slave Message Synchronization

- The Kafka partition replica (follower) pulls stored messages below the high watermark from the leader to the local logs.



- 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 replicas). 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.
- In other words, the follower tries to be in synchronization with the leader by ReplicaFetcherThread.

Kafka HA

- A partition may have multiple replicas (equivalent to `default.replication.factor=N` in the `server.properties` configuration).
- If there are no replicas, once the broker breaks down, all the partition data on the broker cannot be consumed, and the producer cannot store data in the partition.
- After replication is introduced, a partition may have multiple replicas. One of these replicas is elected to act as the leader. Producers and consumers interact only with the leader, and the rest of the replicas act as the followers to copy messages from the leader.

Leader Failover (1)

- 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.
- According to the write process, all replicas in ISR (in-sync replicas) have fully caught up with the leader. Only replicas in ISR can be elected as the leader.
- For $f+1$ replicas, a partition can tolerate the number of f failures without losing committed messages.

- Kafka is not fully synchronous or asynchronous. It is an ISR mechanism: 1. 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. 2. 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. 3. The leader commits messages only when all replicas in ISR send ACK messages to the leader.

Leader Failover (2)

- 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.

- Kafka is not fully synchronous or asynchronous. It is an ISR mechanism: 1. 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. 2. 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. 3. The leader commits messages only when all replicas in ISR send ACK messages to the leader.

Data Reliability

- All Kafka messages are persisted on the disk. When you configure Kafka, you need to set the replication for a topic partition, which ensures data reliability.
- How data reliability is ensured during message delivery?

Message Delivery Semantics

- There are three data delivery modes:
 - At Most Once
 - Messages may be lost.
 - Messages are never redelivered or reprocessed.
 - At Least Once
 - Messages are never lost.
 - Messages may be redelivered and reprocessed.
 - Exactly Once
 - Messages are never lost.
 - Messages are processed only once.

- Exactly Once: has not been achieved yet in Kafka.

Reliability Assurance — Idempotency

- Idempotent operations produce the same result no matter how many times they are performed.
- Principles:
 - 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.

Reliability Assurance — ACK Mechanism

- The producer needs the acknowledgement (ACK) signal sent by the server after receiving the data. This configuration refers to how many ACK signals the producer needs. This configuration actually represents the availability of data backup. The following settings are typical options:
 - **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.
 - **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 leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost provided that at least one ISR remains alive. This is the strongest available guarantee.

Old Data Processing Methods

- 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.

```
-rw----- 1 omm wheel 10485760 Jun 13 13:44 00000000000000000000000000000000.index  
-rw----- 1 omm wheel 1081187 Jun 13 13:45 00000000000000000000000000000000.log
```

- 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.
- Configure the Kafka server properties file:

```
$KAFKA_HOME/config/server.properties
```

- To ensure linear increase of Kafka throughput, a topic is physically divided into one or more partitions and each partition physically corresponds to a directory for storing all messages and index files of the partition.

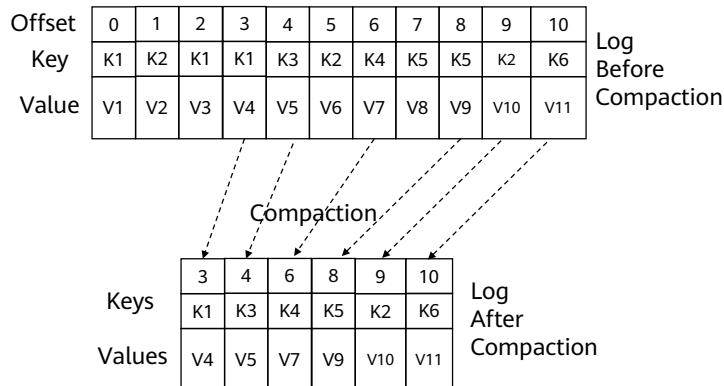
Log Cleanup

- Log cleanup policies: **delete** and **compact**
- Threshold for deleting logs: retention time limit and size of all logs in a partition

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

- 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 limitations, it is impossible to permanently retain all data (actually unnecessary). Therefore, Kafka provides two policies to delete historical data.

Log Compact



- The compact cleanup policy ensures that Kafka will always retain at least the last known value for each message key within the log. For example, a message contains keys and values. If there are three messages whose key is K1, and their values are V1, V3, and V4, respectively. Then only the last message (that is, the message whose value is V4) is retained when the compact policy is adopted. (How to determine which message is the latest message? The message with a larger offset is the latest message that is stored at last as messages are stored in sequence.)
- In other words, only the latest version of data is retained.

Quiz

1. (Single-choice) Each message released to the Kafka cluster belongs to a category called topic. ()
A. True
B. False
2. (Single-choice) In point-to-point messaging systems, messages are persisted to a queue to be consumed by one or more consumers. ()
A. True
B. False

- Answer: 1. A; 2. A

Quiz

3. (Multiple-choice) Which of the following are characteristics of Kafka? ()
 - A. High throughput
 - B. Distributed
 - C. Message persistence
 - D. Random message reading
4. (Single-choice) Which of the following components does the Kafka cluster depend on during its running? ()
 - A. HDFS
 - B. ZooKeeper
 - C. HBase
 - D. Spark

- Answer: 3. ABCD; 4. B

Summary

- This chapter introduces two components, Flume and Kafka. It first describes the functions and application scenarios of Flume and gives details on its basic concepts, features, reliabilities, and configurations. It then describes the basic concepts of the messaging system, and Kafka application scenarios, system architecture, and data management functions.

Acronyms or Abbreviations

- JDBC: Java Database Connectivity
- MRS: MapReduce Service
- WAL: Write Ahead Log
- Syslog: system log
- HTTP: Hypertext Transfer Protocol
- JMS: Java Message Service
- ISR: in-sync replica
- msg: message

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support:
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud:
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

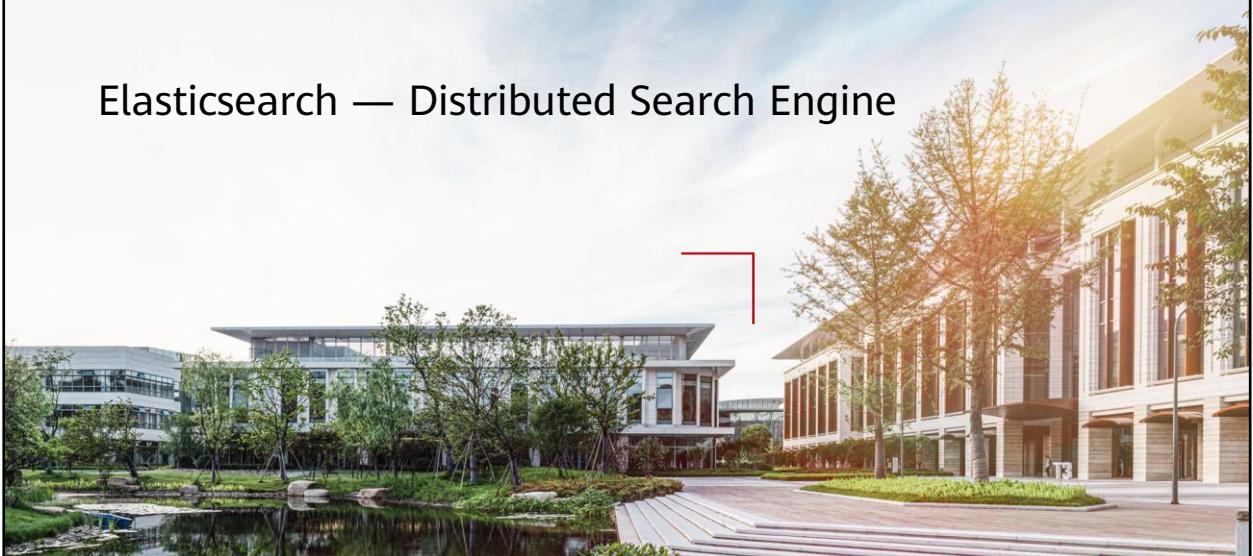
把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



Elasticsearch — Distributed Search Engine



Foreword

- 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.

Objectives

- Upon completion of this course, you will be able to:
 - Know basic functions and concepts of Elasticsearch.
 - Master application scenarios of Elasticsearch.
 - Understand the system architecture of Elasticsearch.
 - Know the key features of Elasticsearch.

Contents

- 1. Overview**
2. System Architecture
3. Key Features

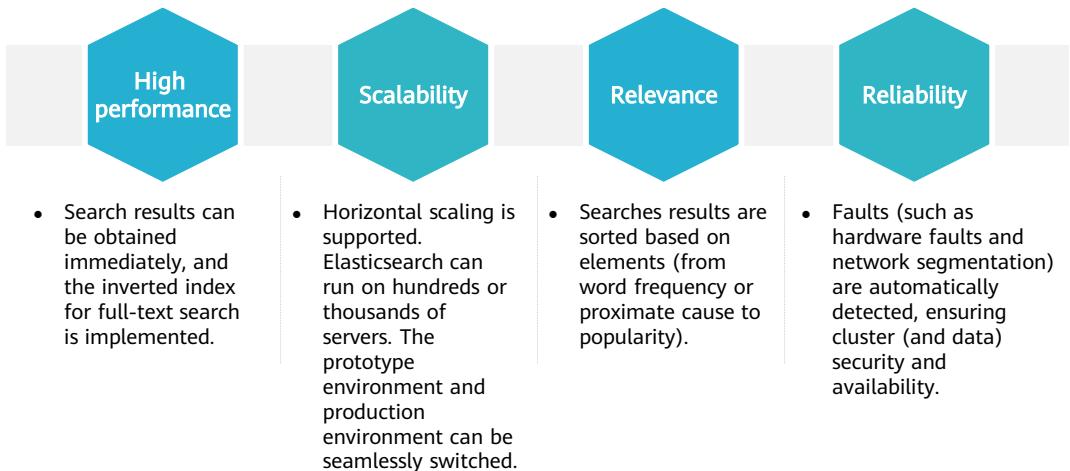
Elasticsearch Overview

- 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.
 - Lucene extension
 - Seamless switchover between the prototype environment and production environment
 - Horizontal scaling
 - Support for structured and unstructured data

- API: Java and RESTful APIs with Lucene as the infrastructure. Local files, shared files, and HDFS are used to store indexes.
- Java Management Extensions (JMX): framework that embeds management functions for applications, devices, and systems. JMX supports development of system, network, and service management applications across heterogeneous OS platforms, system structure, and network transmission protocols.
- Lucene Directory: framework service discovery and master node election of Lucene. ZenDiscovery is used to implement auto node discovery and election of the master node. If the master node is faulty, other nodes are automatically elected, and a new master node will be generated. Plugins: extending the basic functions of Elasticsearch with custom functions, such as custom type mapping, analyzer, local script, and auto discovery. Scripting: used to calculate the value of the custom expression, for example, calculate the correlation score of the custom query. The supported script languages include Groovy, JS, MVEL (deprecated in Elasticsearch 1.3.0), and Python. Discovery: responsible for auto node discovery and master node election in the 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, when a node is added to or removed from the cluster, the master node reallocates shards. River: data source of Elasticsearch and also a method for other storage mode such as a database (the official Rivers are provided for CouchDB, RabbitMQ, Twitter, and Wikipedia, etc) to synchronize data to Elasticsearch. It is an Elasticsearch service that exists as a plugin, which reads data from the River and indexes it to Elasticsearch.
- 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 Simple Storage Service (Amazon S3).
- transport: interaction mode between an Elasticsearch internal node or cluster and the client. By default, internal nodes use the TCP protocol for interaction. In addition, such transmission protocols (integrated using plugins) as the HTTP (JSON format), Thrift, Servlet, Memcached,

and ZeroMQ are also supported.

Elasticsearch Features



- High performance/speed
 - The search result is displayed immediately. We implement the inverted index for full-text search with finite state converters, storage of numerical and geographic location data with BKD trees, and analysis with column storage. Since each piece of data is indexed, there is no need to worry about some data not being indexed.
- Scalability
 - Elasticsearch can run on a laptop. It can also run on hundreds or thousands of servers that carry PB-level data.
 - The prototype environment and production environment can be seamlessly switched. You 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.
- Relevance
 - Elasticsearch searches for all content and finds the required information. Searched results are sorted based on elements (from word frequency, proximate cause to popularity). These content and functions are mixed and matched to adjust the way the results are displayed to the user. Elasticsearch can handle human errors in complex situations (such as spelling errors).
- Reliability
 - Elasticsearch detects faults (hardware faults and network segmentation) for you and ensures the security and availability of your cluster (and data). The cross-cluster replication function enables the hot backup of clusters at any time. Elasticsearch runs in a distributed environment, which is considered at the

beginning of the design, to keep security and reliability.

Elasticsearch Application Scenarios

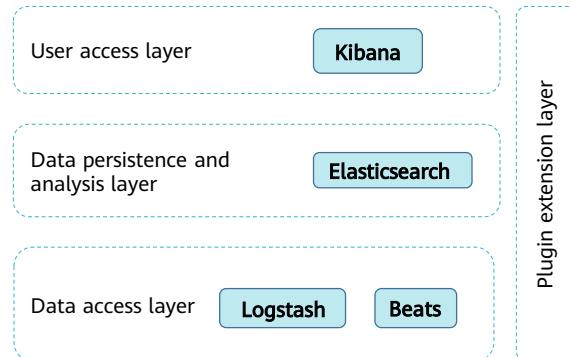
- Elasticsearch is used for log search and analysis, spatiotemporal search, time sequence search, and intelligent search.
 - Complex data types: Structured data, semi-structured data, and unstructured data need to be queried. Elasticsearch can perform a series of operations such as cleansing, word segmentation, and inverted index creation on the preceding data types, and then provide the full-text search capability.
 - Diversified search criteria: Full-text search criteria contain words or phrases.
 - Write and read: The written data can be searched in real time.

- API: Java and RESTful APIs with Lucene as the infrastructure. Local files, shared files, and HDFS are used to store indexes.
- Java Management Extensions (JMX): framework that embeds management functions for applications, devices, and systems. JMX supports development of system, network, and service management applications across heterogeneous OS platforms, system structure, and network transmission protocols.
- Lucene Directory: framework service discovery and master node election of Lucene. ZenDiscovery is used to implement auto node discovery and election of the master node. If the master node is faulty, other nodes are automatically elected, and a new master node will be generated. Plugins: extending the basic functions of Elasticsearch with custom functions, such as custom type mapping, analyzer, local script, and auto discovery. Scripting: used to calculate the value of the custom expression, for example, calculate the correlation score of the custom query. The supported script languages include Groovy, JS, MVEL (deprecated in Elasticsearch 1.3.0), and Python. Discovery: responsible for auto node discovery and master node election in the 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, when a node is added to or removed from the cluster, the master node reallocates shards. River: data source of Elasticsearch and also a method for other storage mode such as a database (the official Rivers are provided for CouchDB, RabbitMQ, Twitter, and Wikipedia, etc) to synchronize data to Elasticsearch. It is an Elasticsearch service that exists as a plugin, which reads data from the River and indexes it to Elasticsearch.
- 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 Simple Storage Service (Amazon S3).
- transport: interaction mode between an Elasticsearch internal node or cluster and the client. By default, internal nodes use the TCP protocol for interaction. In addition, such transmission protocols (integrated using plugins) as the HTTP (JSON format), Thrift, Servlet, Memcached,

and ZeroMQ are also supported.

Elasticsearch Ecosystem

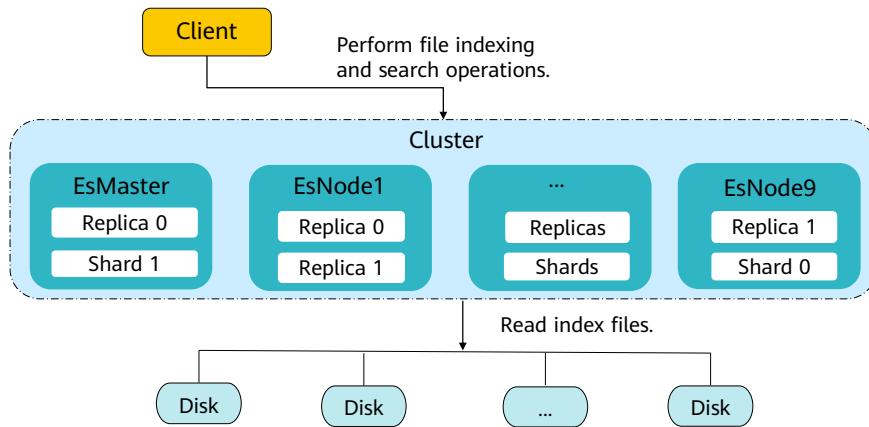
- ELK/ELKB provides a complete set of solutions. They are open-source software and work together to meet diverse requirements.



Contents

1. Overview
- 2. System Architecture**
3. Key Features

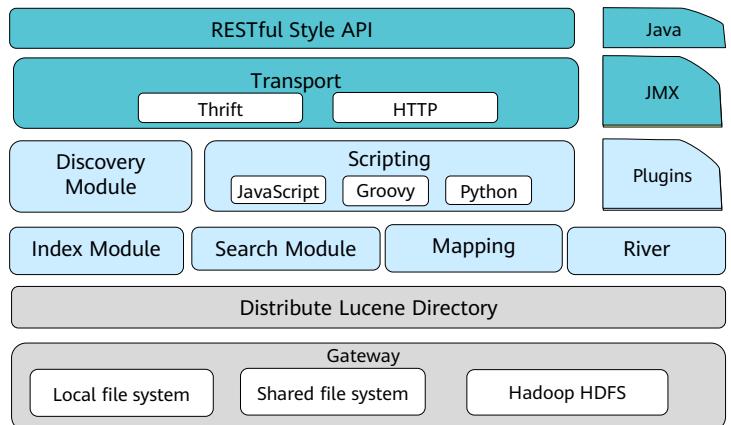
Elasticsearch System Architecture



- 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.
- EsNode: indicates an Elasticsearch node or an Elasticsearch instance.
- EsMaster: indicates an Elasticsearch master node. The master node 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.
- Shard: indicates an index shard. Elasticsearch can split a complete index into multiple shards and distribute them on different nodes.
- Replica: indicates an index replica. Elasticsearch allows you to set multiple replicas for an index. Replicas can improve the fault tolerance of the system. When a shard on a node is damaged or lost, the data can be recovered from the replica. Replicas can also improve the search efficiency of Elasticsearch by automatically balancing the load of search requests.
- High performance: virtual memory design; disk cache parameters; swap tuning; number of replicas; refresh intervals; merge parameters; mapping setting; periodical cache cleaning
- Multi-tenant: The number of Elasticsearch instances is set based on the server memory. Number of Elasticsearch instances on a single node = Available memory of Elasticsearch/64 GB; The number of logical disks on the server must be the same as the number of Elasticsearch instances. Therefore, RAID needs to be configured for the Elasticsearch server based on the number of instances. For example, if the recommended memory is 256 GB and 24 disks are configured, the number of Elasticsearch instances is four. Four RAID groups are required, and each group contains 6 disks.
- ZooKeeper: mandatory in Elasticsearch, storage of security authentication information

Elasticsearch Internal Architecture

- Elasticsearch provides RESTful APIs or APIs for other languages (such as Java).
- The cluster discovery mechanism is used.
- Script languages are supported.
- The underlying layer is based on Lucene, ensuing absolute independence of Lucene.
- Indexes are stored in local files, shared files, and HDFS.



- API: Java and RESTful APIs with Lucence as the infrastructure. Local files, shared files, and HDFS are used to store indexes.
- Java Management Extensions (JMX): framework that embeds management functions for applications, devices, and systems. JMX supports development of system, network, and service management applications across heterogeneous OS platforms, system structure, and network transmission protocols.
- Lucence Directory: framework service discovery and master node election of Lucene. ZenDiscovery is used to implement auto node discovery and election of the master node. If the master node is faulty, other nodes are automatically elected, and a new master node will be generated. Plugins: extending the basic functions of Elasticsearch with custom functions, such as custom type mapping, analyzer, local script, and auto discovery. Scripting: used to calculate the value of the custom expression, for example, calculate the correlation score of the custom query. The supported script languages include Groovy, JS, MVEL (deprecated in Elasticsearch 1.3.0), and Python. Discovery: responsible for auto node discovery and master node election in the 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, when a node is added to or removed from the cluster, the master node reallocates shards. River: data source of Elasticsearch and also a method for other storage mode such as a database (the official Rivers are provided for CouchDB, RabbitMQ, Twitter, and Wikipedia, etc) to synchronize data to Elasticsearch. It is an Elasticsearch service that exists as a plugin, which reads data from the River and indexes it to Elasticsearch.
- 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 Simple Storage Service (Amazon S3).
- transport: interaction mode between an Elasticsearch internal node or cluster and the client. By default, internal nodes use the TCP protocol for interaction. In addition, such transmission protocols (integrated using plugins) as the HTTP (JSON format), Thrift, Servlet, Memcached,

and ZeroMQ are also supported.

Basic Concepts of Elasticsearch (1)

- Index
 - A logical namespace in Elasticsearch
- Type
 - Used to store different types of documents. It is deleted in Elasticsearch 7.
- Document
 - A basic unit that can be indexed
- Mapping
 - Used to restrict the field type

- 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.
- Type: indicates the document type. If documents of various structures are stored in an index, you can find the parameter mapping information according to the document type, facilitating document storage. A type is similar to a table in a database. An index corresponds to a document type. Elasticsearch 6 mentions that Elasticsearch 7 will delete the type. In Elasticsearch 6, each index can have only one type. In Elasticsearch 7, the default _doc is used as the type. According to the official statement, the type will be removed in Elasticsearch 8.x.
- Document: A document is a basic unit of information that can be indexed. It refers to JSON data at the top-level structure or obtained by serializing the root object. A document is similar to a row in a 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.

Basic Concepts of Elasticsearch (2)

- Cluster
 - Each cluster contains multiple nodes, one of which is the master node (the rest are slave nodes). The master node can be elected.
- EsNode
 - Elasticsearch node. A node is an Elasticsearch instance.
- EsMaster
 - Master node that 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.
- shards
 - Index shard. Elasticsearch splits a complete index into multiple shards and distributes them on different nodes.

- 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.
- EsNode: indicates an Elasticsearch node or an Elasticsearch instance.
- EsMaster: indicates an Elasticsearch master node. The master node 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.
- Shard: indicates an index shard. Elasticsearch can split a complete index into multiple shards and distribute them on different nodes.
- Replica: indicates an index replica. Elasticsearch allows you to set multiple replicas for an index. Replicas can improve the fault tolerance of the system. When a shard on a node is damaged or lost, the data can be recovered from the replica. Replicas can also improve the search efficiency of Elasticsearch by automatically balancing the load of search requests.
- Recovery: data recovery or re-distribution. When a node is added to or deleted from the cluster, Elasticsearch redistributes shards based on the load of the node. When a failed node is restarted, data will be recovered.

Basic Concepts of Elasticsearch (3)

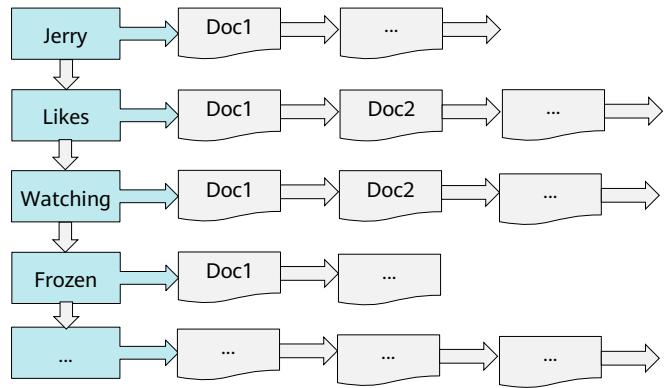
- Replicas
 - Index replica. Elasticsearch allows you to set multiple replicas for an index. Replicas can improve the fault tolerance of the system. When a shard on a node is damaged or lost, the data can be recovered from the replica. In addition, replicas can improve the search efficiency of Elasticsearch by automatically balancing the load of search requests.
- Recovery
 - Data recovery or re-distribution. When a node is added to or deleted from the cluster, Elasticsearch redistributes shards based on the load of the node. When a failed node is restarted, data will be recovered.
- 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, and 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.
- Transport
 - 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.

Contents

1. Overview
2. System Architecture
- 3. Key Features**

Elasticsearch Inverted Index

- Forward index: Values are searched for based on keys.
That is, specific information that meets the search criteria is located based on keys.
- Inverted index: Keys are searched for based on values. In full-text search, a value is the searched keyword.
Corresponding documents are located based on values.



- The traditional search mode (forward index) is based on key points. That is, search for keywords based on document numbers.
- Elasticsearch (Lucene) uses the inverted index mode. 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. For example, an inverted index means querying the document number using the keyword and then locating the document based on the document number. This is similar to looking up a dictionary or querying the content of a book to locate the content of a specified page.

Elasticsearch Access APIs

- Elasticsearch can initiate RESTful requests to operate data. The request methods include GET, POST, PUT, DELETE, and HEAD, which allow you to add, delete, modify, and query documents and indexes.
- For example, use Kibana to execute RESTful APIs.

```
1. View the cluster health status.  
GET /_cat/health?v&pretty  
2. Create a small index with only one primary shard and no replicas.
```

```
PUT /my_temp_index  
{  
  "settings": {  
    "number_of_shards" : 1,  
    "number_of_replicas" : 0  
  }  
}
```

```
3. Delete multiple indexes.
```

```
DELETE /index_one,index_two  
4. Add documents by automatically generated IDs.  
POST /person/man  
{  
  "name":"111",  
  "age":11  
}
```

- For example, access Elasticsearch using the cURL client.

```
Obtain the cluster health status.  
curl -XGET "http://ip:port/_cluster/health?pretty"
```

- If you add **?pretty** after a query statement, generated JSON is clearer and much easier to read.

Elasticsearch Routing Algorithm

- Elasticsearch provides two routing algorithms:
 - Default route: $\text{shard} = \text{hash}(\text{routing}) \% \text{number_of_primary_shards}$. 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.

Elasticsearch Balancing Algorithm

- Elasticsearch provides the automatic balancing function.
- Application scenarios: capacity expansion, capacity reduction, and data import
- The algorithms are as follows:
 - $\text{weight_index}(\text{node}, \text{index}) = \text{indexBalance} * (\text{node.numShards(index)} - \text{avgShardsPerNode(index)})$
 - $\text{Weight_node}(\text{node}, \text{index}) = \text{shardBalance} * (\text{node.numShards()} - \text{avgShardsPerNode})$
 - $\text{weight}(\text{node}, \text{index}) = \text{weight_index}(\text{node}, \text{index}) + \text{weight_node}(\text{node}, \text{index})$

Elasticsearch Capacity Expansion

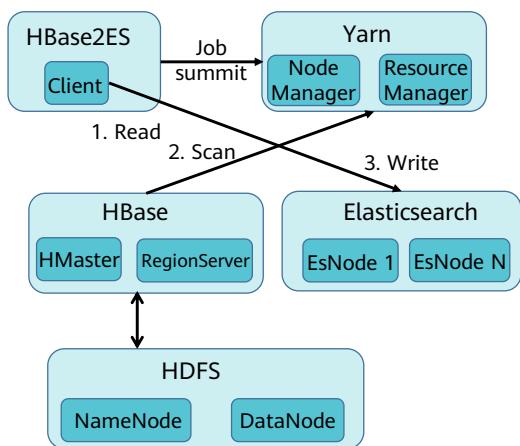
- Scenarios:
 - High physical resource consumption such as high CPU and memory usages of Elasticsearch service nodes, and insufficient disk space
 - Excessive index data volume for one Elasticsearch instance, such as 1 billion data records or 1 TB data
- Methods:
 - Add EsNode instances.
 - Add nodes with EsNode instances.
- After capacity expansion, use the automatic balancing policy.

Elasticsearch Capacity Reduction

- Scenarios:
 - OS reinstallation on nodes required
 - Reduced amount of cluster data
 - Out-of-service
- Method:
 - Delete an Elasticsearch instance on the Cloud Search Service (CSS) console.
- Precautions:
 - Ensure that replicas in the shard of the instance to be deleted exist in another instance.
 - Ensure that data in the shard of the instance to be deleted has been migrated to another node.

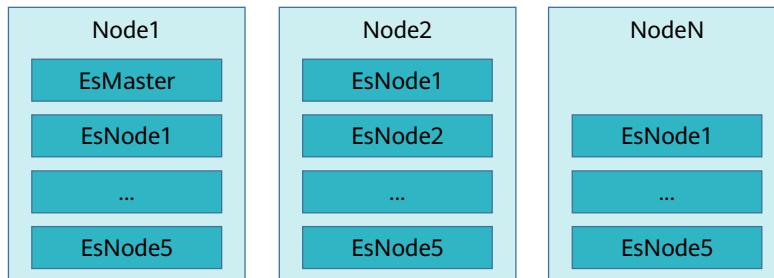
Elasticsearch Indexing HBase Data

- When HBase data is written to Elasticsearch, 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.
- Batch indexing: 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.



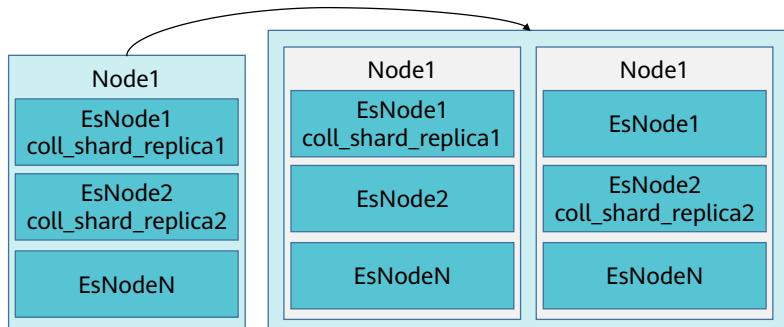
Elasticsearch Multi-Instance Deployment on a Node

- 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.



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.



Other Elasticsearch Features

- HBase full-text indexing
 - After the HBase table and Elasticsearch indexes are mapped, indexes and raw data can be stored in Elasticsearch and HBase, respectively. The HBase2ES tool is used for offline indexing.
- Encryption and authentication
 - Encryption and authentication are supported for a user to access Elasticsearch in a security cluster.

Quiz

1. ____ is the basic unit that can be indexed in Elasticsearch.
2. Which type of data can be indexed using Elasticsearch? ()
 - A. Structured data
 - B. Unstructured data
 - C. Semi-structured data
 - D. All of the above

- Answers:
 - 1. Document
 - 2. BD

Quiz

3. (Single-choice) Which of the following is not an application scenario of the Elasticsearch balancing algorithm? ()
- A. Capacity expansion
 - B. Capacity reduction
 - C. Data import
 - D. Data cleansing

- Answers:
 - 3. D

Summary

- This chapter describes the basic concepts, functions, application scenarios, architecture, and key features of Elasticsearch. Understanding the key concepts and features of Elasticsearch allows you to better develop and use the component.

Acronyms and Abbreviations

- JMX: Java Management Extensions
- NoSQL: Not Only SQL, non-relational database
- ELK: Elasticsearch, Logstash, and Kibana
- Es: Elasticsearch
- HTTP: Hypertext Transfer Protocol
- Js: JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions.
- Doc: document
- CPU: Central Processing Unit
- ELKB: Elasticsearch, Logstash, Kibana, and Beats

Recommendations

- Huawei Talent
 - <https://e.huawei.com/en/talent>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>
- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.



MRS: Huawei's Big Data Platform



 HUAWEI

Foreword

- This chapter first provides an overview of Huawei's big data platform, MRS, before looking at its advantages and application scenarios. Then, it describes some MRS components, including Hudi, HetuEngine, Ranger, and LDAP+Kerberos authentication. Finally, it illustrates the MRS cloud-native data lake baseline solution.

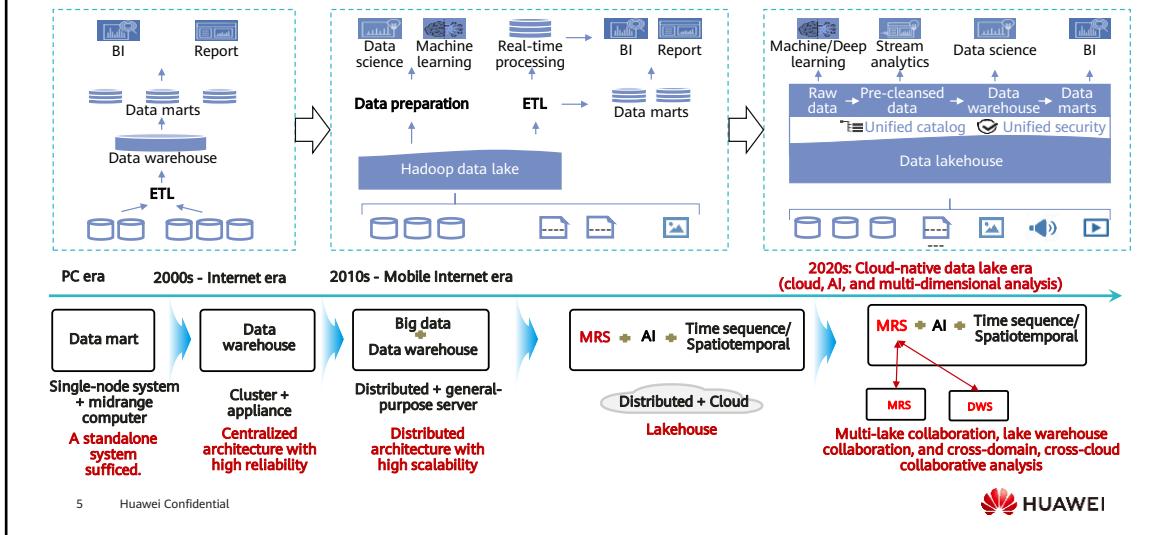
Objectives

- Upon completion of this course, you will possess a deeper understanding of:
 - Huawei's big data platform, MRS
 - Hudi, HetuEngine, Ranger, and LDAP+Kerberos security authentication
 - MRS cloud-native data lake baseline solution

Contents

- 1. Overview of MRS**
2. MRS Components
3. MRS Cloud-Native Data Lake Baseline Solution

Trends in the Evolution of Big Data Technology



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.

Huawei Cloud MRS

- MapReduce Service (MRS) is used to deploy and manage Hadoop systems on Huawei Cloud.
- 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.

- MRS provides Hadoop-based high-performance big data components, such as Hudi, ClickHouse, Spark, Flink, Kafka, and HBase, to support data lakes, data warehouses, business intelligence (BI), AI, and more. MRS supports both hybrid cloud and public cloud deployments. In hybrid cloud deployment, one cloud-native architecture implements offline, real-time, and logical data lakes, helping customers carry out intelligent enterprise upgrade. In public cloud deployment, with MRS, customers can quickly build a low-cost, flexible, open, secure, and reliable one-stop big data platform.

MRS Highlights



Decoupled storage and compute

- A unified data lake eliminates data silos; a single copy of data is enough, with no need for data transfer; multiple computing engines, making flexible allocation and on-demand scaling of storage and computing resources possible; 30% more cost-effective than the industry average



Cutting-edge open-source technologies

- In-depth reconstruction of mainstream engines, such as Spark, Hive, and Flink, with key technologies such as indexing, caching, and metadata; Huawei-developed CarbonData with millisecond-level point query; Superior Scheduler for 20,000+ nodes in a single cluster



First-class performance and experience

- Full-stack performance acceleration and millisecond-level response to millions of metadata records by means of four-level vertical optimization on hardware, data organization, computing engine, and AI intelligent optimization, providing users with the ultimate performance experience

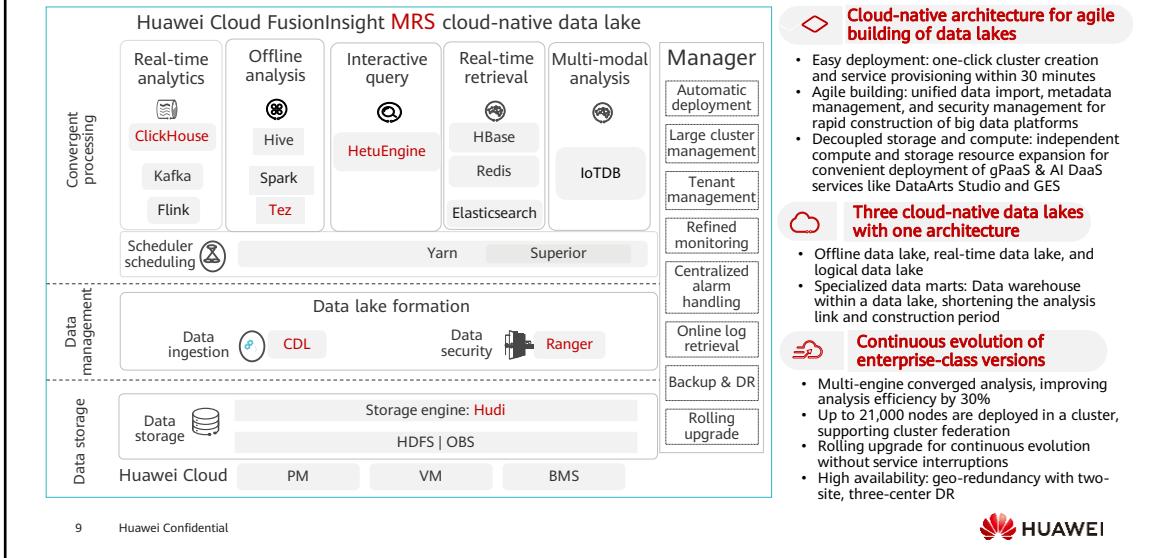


High security and availability

- Cross-AZ HA of a single cluster, eliminating single points of failure (SPOFs), rolling patch installation/upgrade, task reconnection upon disconnection, and zero service interruption; multi-level security assurance capabilities, such as network resource isolation, account security, and data security control

- 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 decoupling of storage and compute resources, greatly improving the resource utilization of big data clusters.
 - MRS supports Huawei-developed CarbonData and Superior Scheduler, delivering better performance.
 - 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.

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.

MRS Application Scenarios



Offline data lake

- High-performance interactive query engine enables data processing within the lake.
- Unified metadata, global data visualization
- Converged analysis and unified SQL query are supported.



Real-time data lake

- High timeliness: Real-time incremental data import to the lake within seconds, from T+1 to T=0
- High resource utilization: Incremental data is scattered into the lake, meaning 2x more enhanced resource utilization
- Stream-batch convergence with unified SQL interfaces



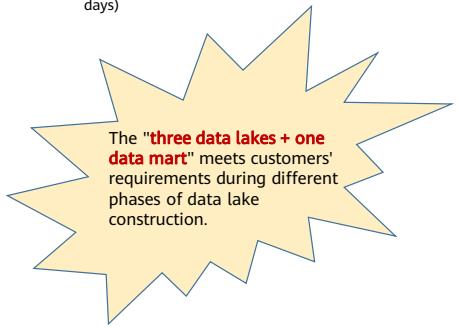
Abundant specialized data marts

- Lakes and marts in the same cluster for unified management and seamless interconnection
- Full self-service, millisecond-level real-time OLAP analysis of ClickHouse
- High-throughput, low-latency time series database IoTDB

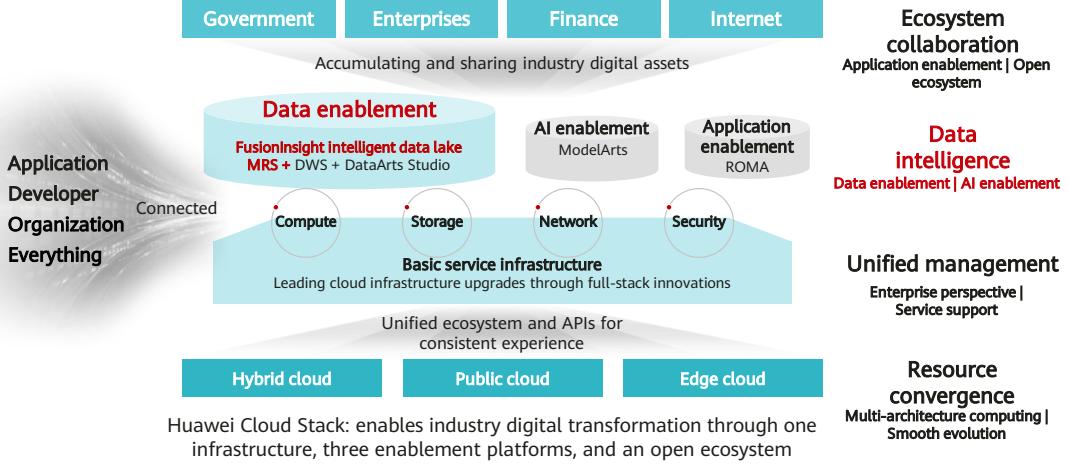


Logical data lake

- Cross-lake, cross-domain, cross-warehouse, and all-domain data collaborative analysis
- Reduced data migration for computing without moving data, 50x higher analysis efficiency
- 10x faster service rollout efficiency (weeks -> days)



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



Contents

1. Overview of MRS
2. Components
 - Hudi
 - HetuEngine
 - Ranger
 - LDAP+Kerberos Security Authentication
3. MRS Cloud-Native Data Lake Solution

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 in a data lake table format, which provides the ability to update and delete data as well as consume new data on HDFS. It supports multiple compute engines and provides insert, update, and delete (IUD) interfaces and streaming primitives, including upsert and incremental pull, over datasets on HDFS.
- 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.

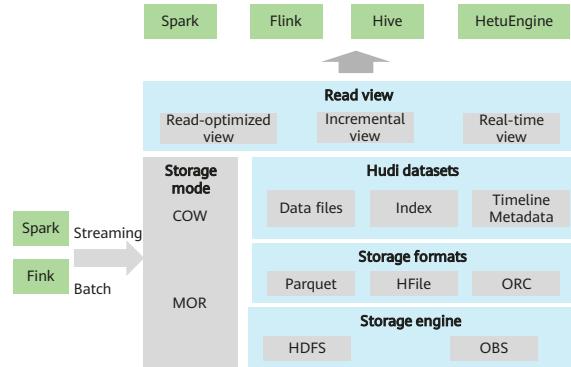
Hudi Features

- Supports fast updates through custom indexes.
- Supports snapshot isolation for data write and query.
- Manages file size and layout based on statistics.
- Supports timeline.
- Supports data rollback.
- Supports savepoints for data restoration.
- Merges data asynchronously.
- Optimizes data lake storage using the clustering mechanism.

- 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.

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

- Storage modes
 - Copy On Write (COW): high read performance and slower write speed than that of MOR
 - Merge on Read (MOR): high write performance and lower read performance
- Storage formats
 - The open-source Parquet and HFile formats are supported. The support for ORC is under planning.
- Storage engines
 - Open-source HDFS and Huawei Cloud Object Storage Service (OBS)
- Views
 - Read-optimized view
 - Incremental view
 - Real-time view



15 Huawei Confidential



- 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.

Contents

1. Overview of MRS

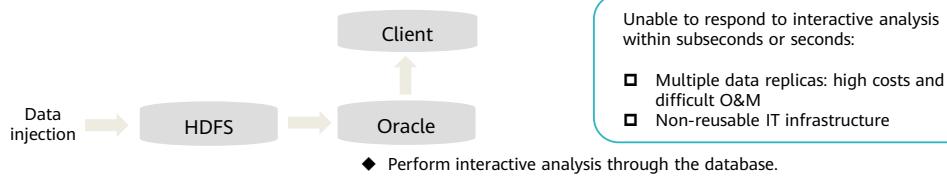
2. Components

- Hudi
- HetuEngine
- Ranger
- LDAP+Kerberos Security Authentication

3. MRS Cloud-Native Data Lake Baseline Solution

A Big Data Ecosystem Requires Interactive Query and Unified SQL Access

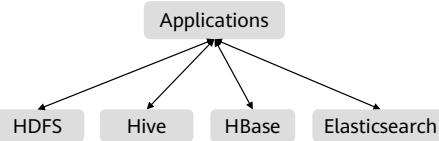
- Services require interactive analysis within subseconds or seconds, which may result in redundant data replicas.



Unable to respond to interactive analysis within subseconds or seconds:

- Multiple data replicas: high costs and difficult O&M
- Non-reusable IT infrastructure

- Unified SQL access is required due to diversified components in the data lake.

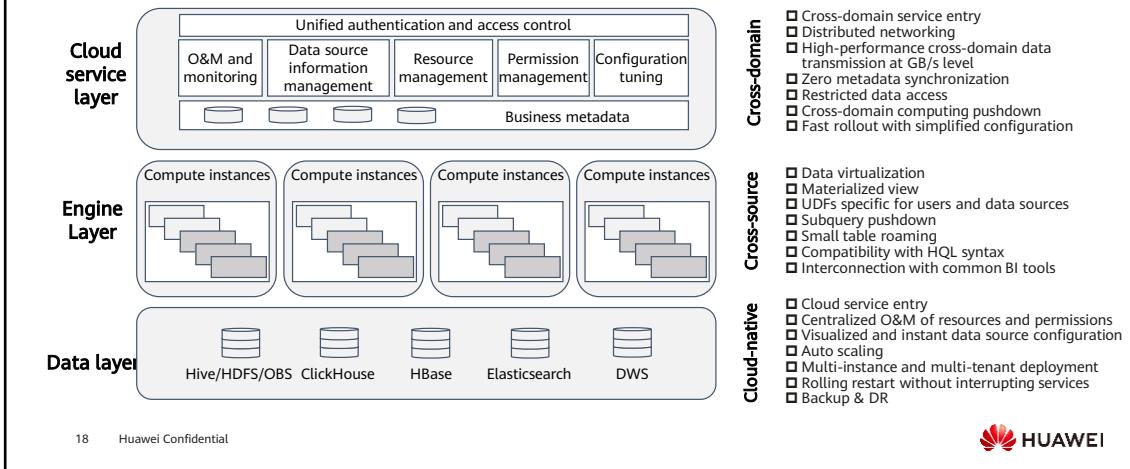


Diversified data components:

- High costs of service system interconnection and management
- Difficulty in converged data analysis

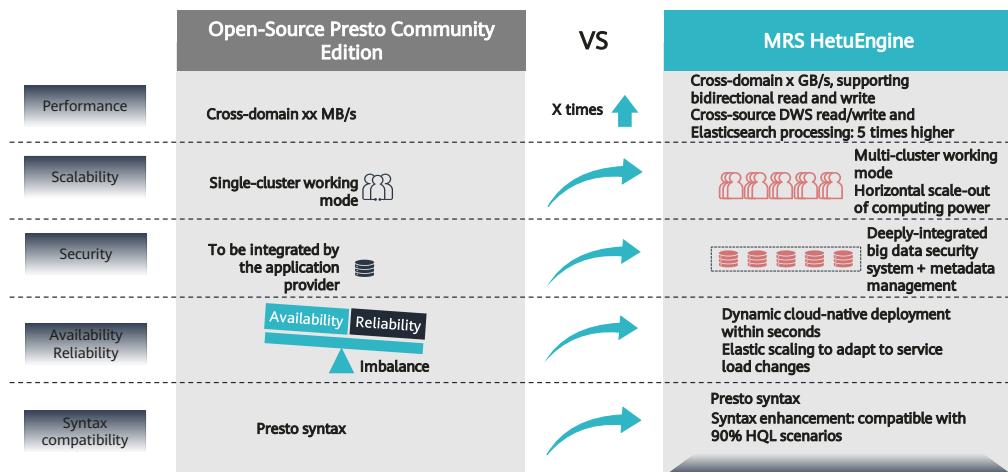
HetuEngine

- HetuEngine is a Huawei-developed high-performance engine for distributed SQL query and data virtualization. Fully compatible with the big data ecosystem, HetuEngine implements mass data query within seconds. It supports heterogeneous data sources, enabling one-stop SQL analysis in the data lake.



- 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 in 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.
- 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).

Open-Source Community Edition vs. HetuEngine



Contents

1. Overview of MRS

2. Components

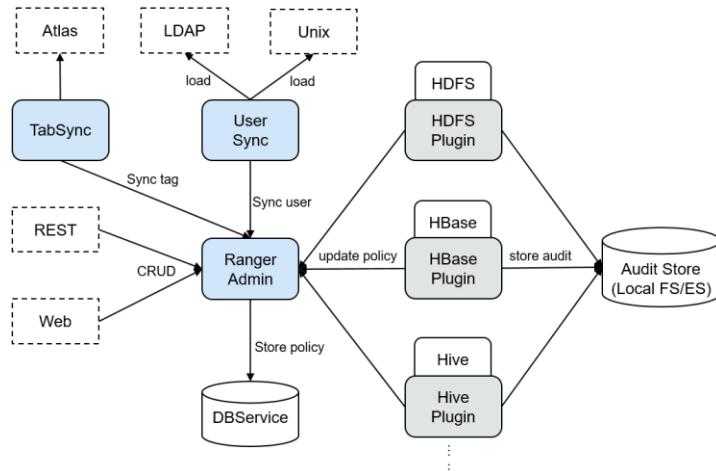
- Hudi
- HetuEngine
- **Ranger**
- LDAP+Kerberos Security Authentication

3. MRS Cloud-Native Data Lake Baseline Solution

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.

Ranger Architecture



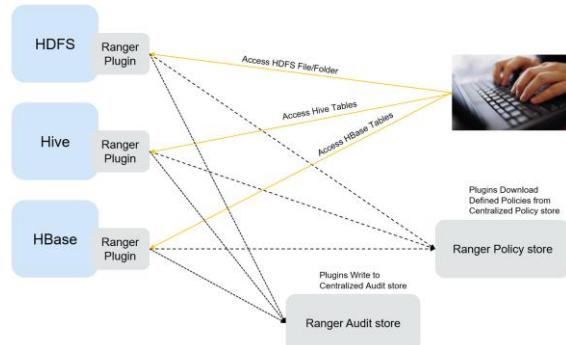
22 Huawei Confidential



- **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.

Relationship Between Ranger and Other Components

- Ranger provides PBAC authentication plug-ins for component servers. Currently, components like HDFS, Yarn, Hive, HBase, Kafka, Storm, and Spark2x support Ranger authentication. More components will become available in the future.



Contents

1. Overview of MRS

2. Components

- Hudi
- HetuEngine
- Ranger
- LDAP+Kerberos Security Authentication

3. MRS Cloud-Native Data Lake Baseline Solution

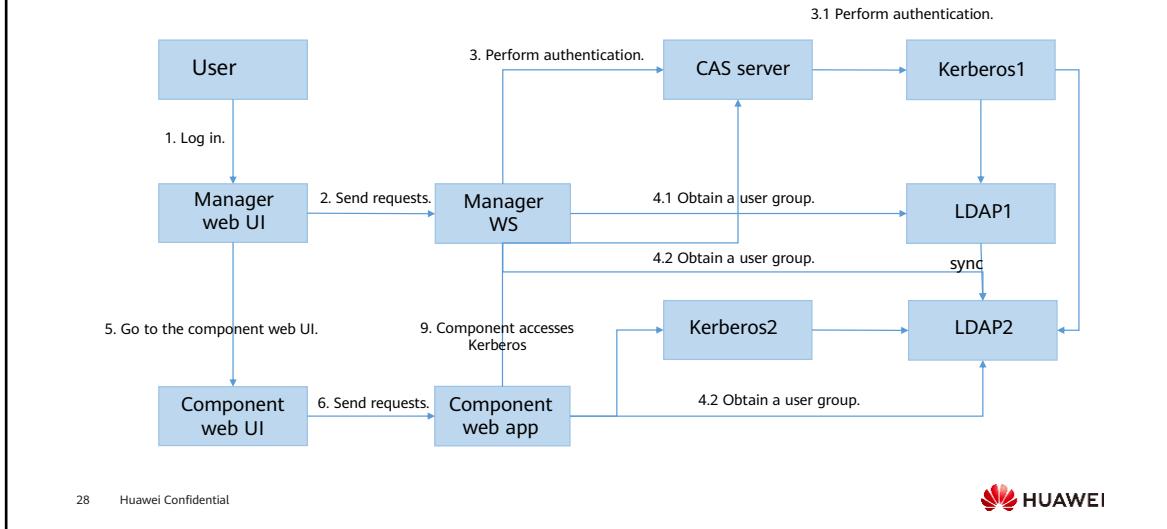
LDAP

- LDAP stands for Lightweight Directory Access Protocol. It is a protocol for implementing centralized account management architecture based on X.500 protocols.
- On the Huawei big data platform, an LDAP server functions as a directory service system to implement centralized account management.
- LDAP has the following characteristics:
 - LDAP runs over TCP/IP or other connection-oriented transfer services.
 - 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*.

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.
- Huawei big data platform uses KrbServers to provide Kerberos functions for all components. To manage access control permissions on data and resources in a cluster, it is recommended that the cluster be installed in security mode. In security mode, a client application must be authenticated and a secure session must be established before the application can access resources in the cluster. MRS uses KrbServers to provide Kerberos authentication for all components, implementing a reliable authentication mechanism.

Architecture of Huawei Big Data Security Authentication Scenarios



- Steps 1, 2, 3, 4:
 - Process for logging in to the Manager web UI.
- Steps 5, 6, 7, 8:
 - Process for logging in to the component UI.
- Step 9:
 - Access between components
- Kerberos1 operations on LDAP data:
 - Active and standby instances of LDAP1 and LDAP2 are accessed in load sharing mode. Data can be written only in the active LDAP2 instance. Data can be read on LDAP1 or LDAP2.
- Kerberos2 operations on LDAP data:
 - Only the active and standby instances of LDAP2 can be accessed. Data can be written only in the active LDAP2 instance.

Enhanced Open-Source LDAP+Kerberos Features

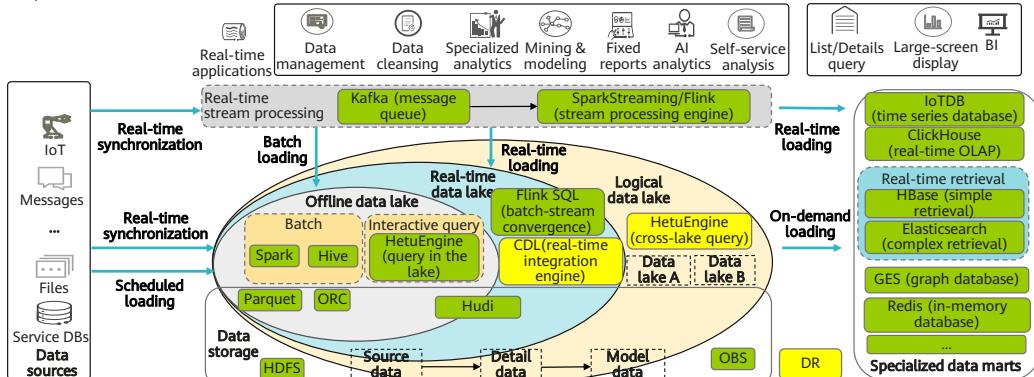
- Service Authentication in the Cluster
 - In an MRS cluster in security mode, mutual access between services is implemented based on the Kerberos security architecture. When a service (such as HDFS) in the cluster is set to start, the corresponding sessionkey (keytab, used for identity authentication of the application) is obtained from Kerberos. If another service (such as Yarn) needs to access HDFS to add, delete, modify, or query data in HDFS, the corresponding TGT and ST must be obtained for secure access.
- Application Development Authentication
 - MRS components provide application development interfaces for customers and upper-layer service product clusters. During application development, a cluster in security mode provides specified application development authentication interfaces to implement application security authentication and access.
- Cross-Manager Mutual Trust
 - MRS provides the mutual trust function between two Managers to implement data read and write operations between them.

Contents

1. Overview of MRS
2. Components
- 3. MRS Cloud-Native Data Lake Baseline Solution**

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

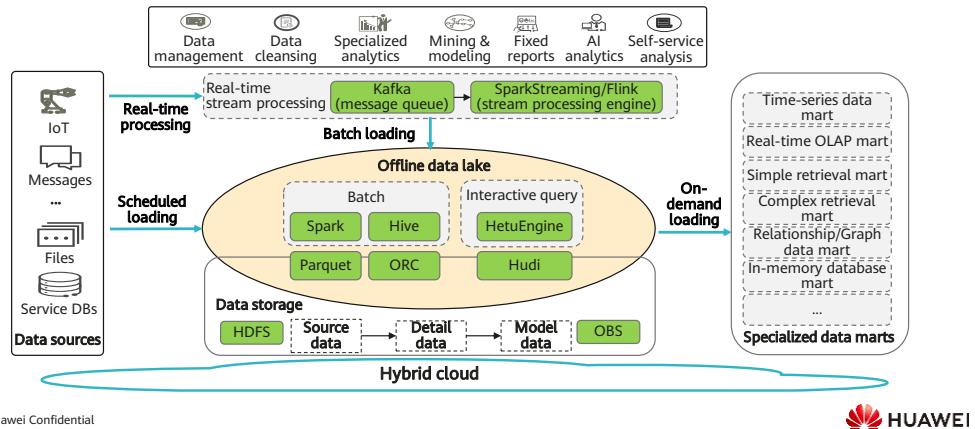
- The MRS data lake solution implements the "three lakes + mart" service scenario to meet customers' requirements in different phases of data lake construction.



Offline Data Lake

Data lake: A big data platform holding a vast amount of data in its native format for an enterprise. Access to data and compute power is granted to users through strict data permission and resource control. In a data lake, one replica of data supports multidimensional analysis.

Offline: Typically, data is not stored in a data lake until a delay of over 15 minutes after being generated, during which period the data is offline.



32 Huawei Confidential



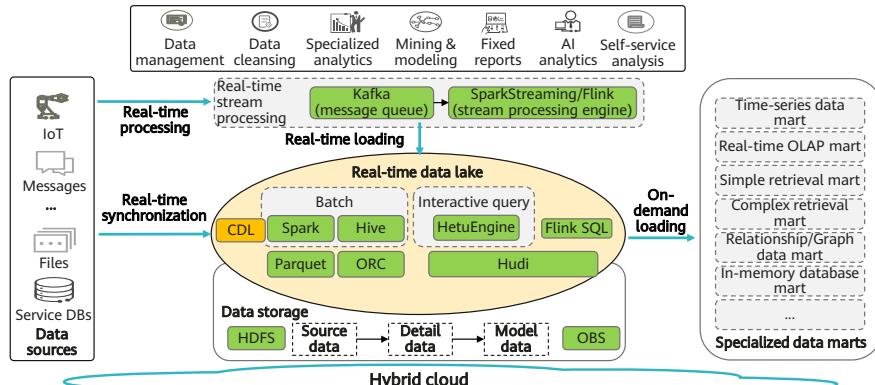
- Q&A:
- Q: What are the differences between an offline data lake and a traditional data warehouse?
- A: Offline data lakes are also called lakehouses in many vendors. A data warehouse is a concept in the database era. It refers to an offline data lake that supports only SQL. The offline data lake has stronger capabilities than the data warehouse. In addition to SQL, the offline data lake supports data mining, AI analytics, offline analytics, and interactive query.
- Q: As the data source, the database is an offline data lake. As the data source, the streaming data is a real-time data lake. Is this correct?
- A: 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.
- Q: Can the offline data lake support human-machine interaction query within seconds?
- A: 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.
- Q: What is the relationship between the offline data lake and the original baseline solution?
- A: Offline data lake = Original batch processing solution + Original real-time

stream processing solution + Original interactive query solution

Real-Time Data Lake

Data lake: A big data platform holding a vast amount of data in its native format for an enterprise. Access to data and compute power is granted to users through strict data permission and resource control. In a data lake, one replica of data supports multidimensional analysis.

Real-time: Real-time refers to cases where data can be stored in the data lake within one minute after being generated, while quasi-real-time is where data is stored in the data lake within 1 to 15 minutes.



33 Huawei Confidential



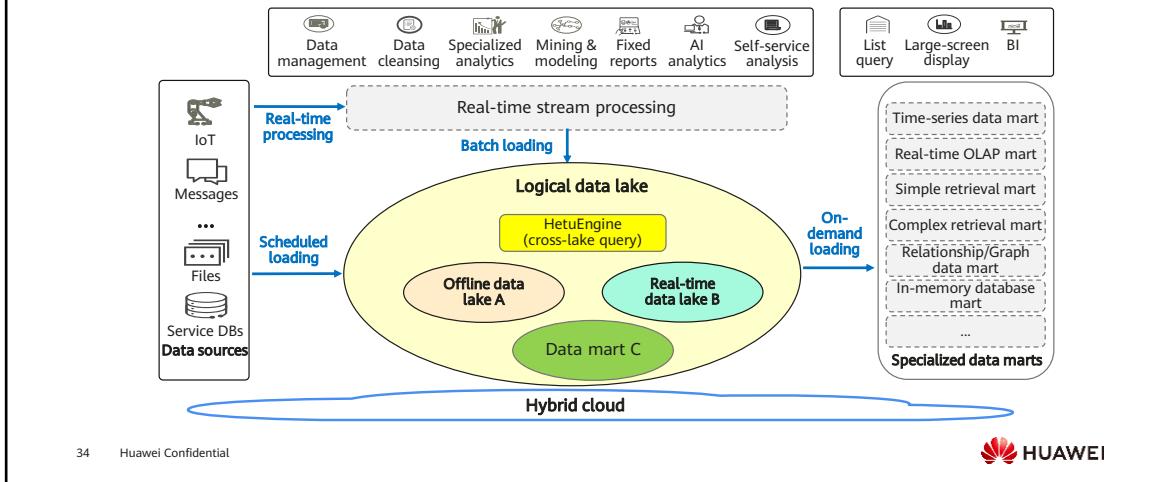
- Q&A
- Q: Are the real-time data lake and offline data lake two data lakes?
- A: The real-time data lake is an upgrade of the offline data lake. It adds the real-time data ingestion capability to the offline data lake and supports the batch-stream integration engine. The two are the same data lake. Some vendors call it "real-time data warehouse".
- Q: As the data source, the database is an offline data lake. As the data source, the streaming data is a real-time data lake. Is this correct?
- A: 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.
- Q: The real-time data lake supports real-time data import, second-level data analysis, and data sharing. Is a specialized data mart required?
- A: 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.
- Q: What is the relationship between the real-time data lake and the original baseline solution?
- A: 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

Logical Data Lake

Data lake: a big data platform holding a vast amount of data in various formats in an enterprise. It opens data and compute power to users with strict data permission and resource control.

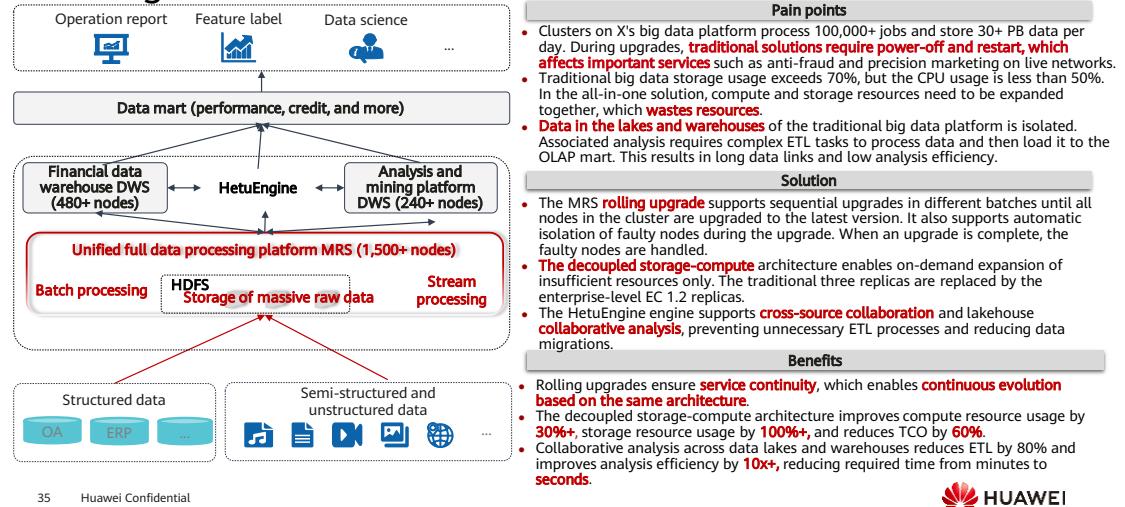
Logical data lake: a virtual data lake composed of multiple physically dispersed data platforms.



34 Huawei Confidential

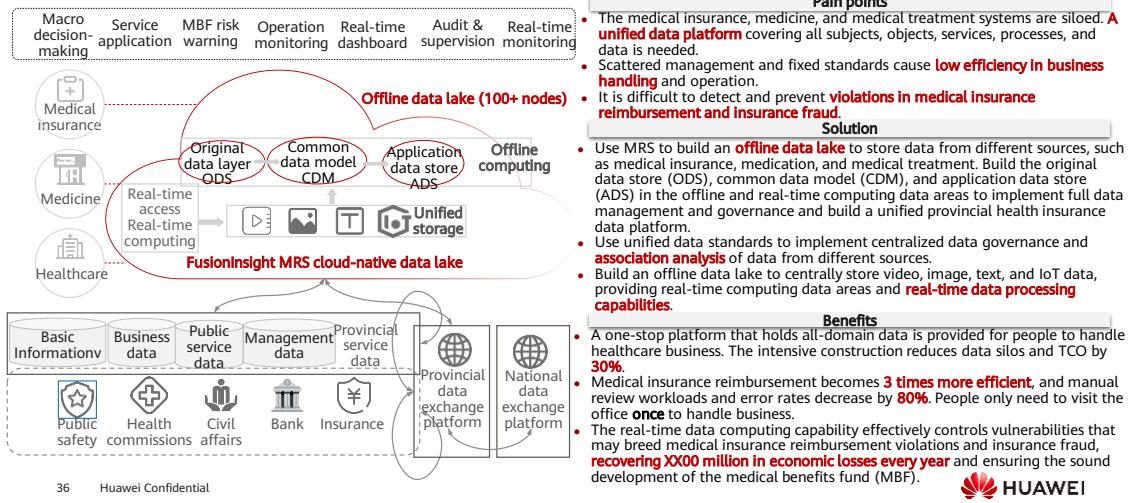
- Q&A
- Q: Is it necessary to build offline or real-time data lakes if there is a logical data lake?
- A: A logical data lake means that data is scattered on multiple data platforms. The timeliness of joint analysis is far worse than that of an offline or real-time data lake. Therefore, a logical data 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).
- Q: Is it necessary to migrate data between multiple offline or real-time data lakes if there is a logical data lake?
- A: 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".
- Q: Is data secure if a logical data lake connects multiple offline or real-time data lakes?
- A: 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.

X Bank: Rolling Upgrades, Decoupled Storage-Compute, and HetuEngine-based Real-Time BI



- 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.

XX Healthcare Security Administration Built a Unified Offline Data Lake for Decision-Making



- XX Healthcare Security Administration built a unified offline data lake for decision-making support.
- Customer's 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.

- 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%.
- Medical insurance reimbursement becomes 3 times more efficient, and manual review workloads and error rates decrease by 80%. People only need to visit the office once to handle business.
- The real-time data computing capability effectively controls vulnerabilities that may breed medical insurance reimbursement violations and insurance fraud, recovering XX00 million in economic losses every year and ensuring sound development of the medical benefits fund (MBF).

Quiz

1. What is MRS?
2. What are the advantages of MRS compared with self-built Hadoop?

- Answers:
 1. MapReduce Service (MRS) is used to deploy and manage Hadoop systems on Huawei Cloud.
 2. 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 decoupling of storage and compute resources, greatly improving the resource utilization of big data clusters. MRS supports Huawei-developed CarbonData and Superior Scheduler, delivering better performance. 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.

Summary

- This chapter first described Huawei's big data platform, MRS, along with its advantages and application scenarios. It then went through some MRS components, including Hudi, HetuEngine, Ranger, and LDAP+Kerberos security authentication. Finally, this chapter introduced the MRS cloud-native data lake baseline solution.

Acronyms and Abbreviations

- BI: Business Intelligence
- ETL: Extract, Transform, and Load, a process that involves extracting data, transforming the data, and loading the data to final targets.
- AI: Artificial Intelligence
- DWS: Data Warehouse Service
- ES: Elasticsearch, distributed full-text search service
- OBS: Object Storage Service
- ORC: OptimizedRC File. ORC is a top-level Apache project and is a self-describing column-based storage.

Acronyms and Abbreviations

- COW: Copy On Write
- MOR: Merge On Read
- UDF: User-Defined Functions
- TCO: Total Cost of Ownership
- ODS: Operational Data Store
- CDM: Cloud Data Migration
- ADS: Anti-DDoS Service

Recommendations

- Huawei Cloud
 - <https://www.huaweicloud.com/intl/en-us/>
- Huawei Talent
 - <https://e.huawei.com/en/talent/portal/#/>
- Huawei Enterprise Product & Service Support
 - <https://support.huawei.com/enterprise/en/index.html>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

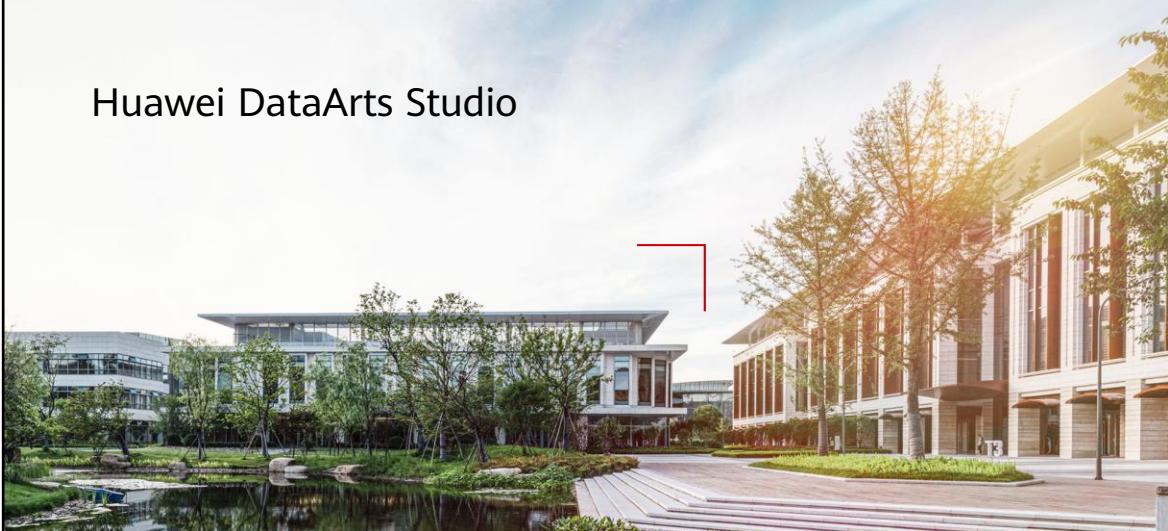
Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive
statements including, without limitation, statements regarding
the future financial and operating results, future product
portfolio, new technology, etc. There are a number of factors that
could cause actual results and developments to differ materially
from those expressed or implied in the predictive statements.
Therefore, such information is provided for reference purpose
only and constitutes neither an offer nor an acceptance. Huawei
may change the information at any time without notice.



Huawei DataArts Studio



Foreword

- This chapter introduces data governance and DataArts Studio (Huawei's data governance service).

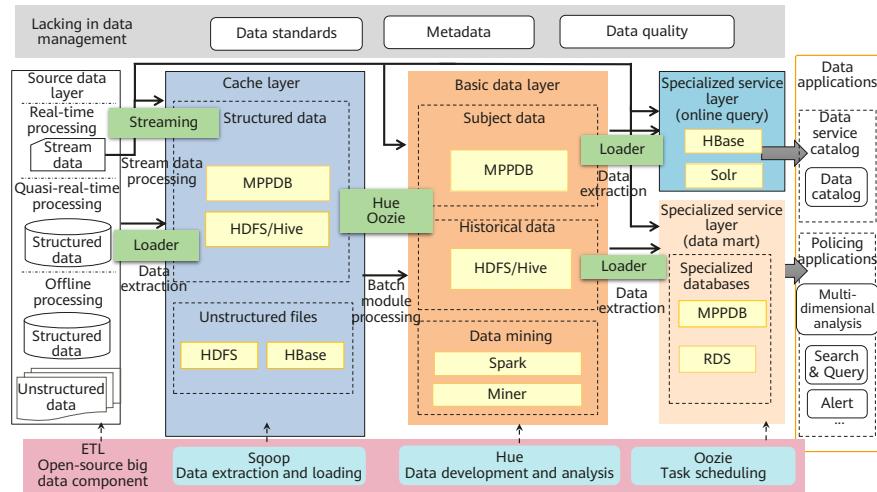
Objectives

- Upon completion of this course, you will understand:
 - Data governance
 - Huawei DataArts Studio

Contents

- 1. Data Governance**
2. Huawei DataArts Studio

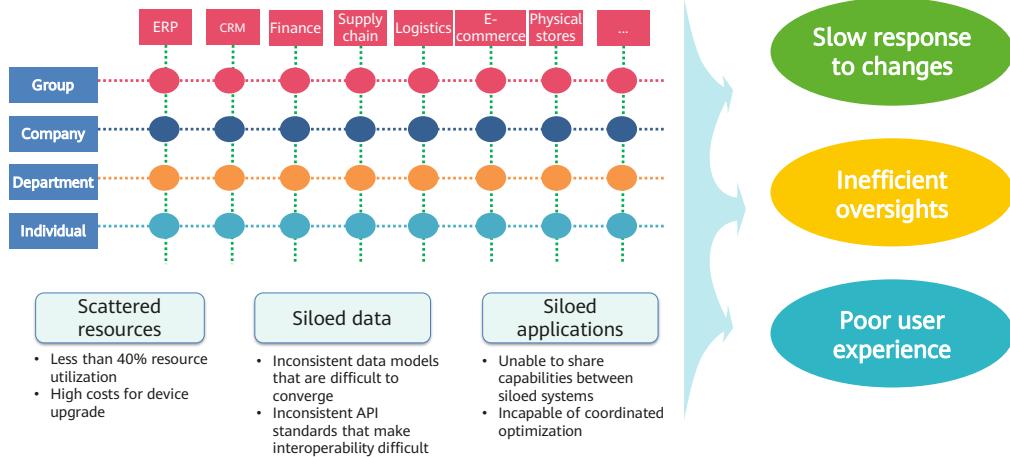
Major Challenges Faced by Big Data O&M Engineers



Challenges

- A large number of big data components involved in the development process.
- A massive number of command lines, SQL scripts, and shell scripts that spread across separate hosts and systems without unified management.
- Oozie lacks graphical orchestration capabilities and does not apply to complex scenarios.
- Open-source big data components lack data management capabilities, creating great difficulty for data governance.

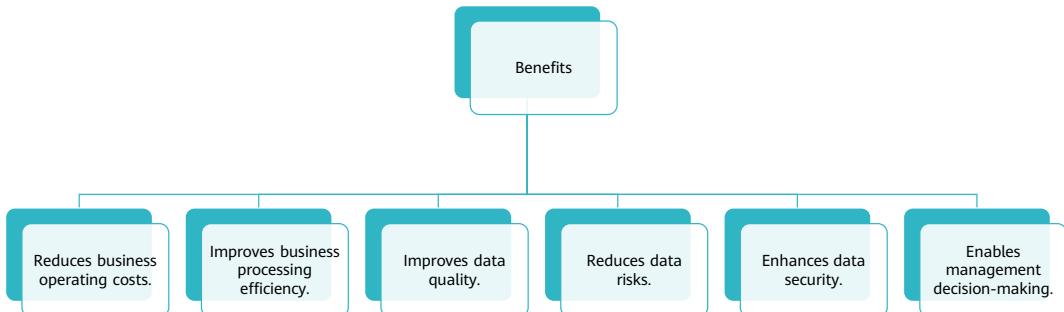
Application and Information Silos Severely Hinder Enterprises' Digital Development



Data Governance

- The discipline of data governance treats data as enterprise assets and utilizes, protects, and optimizes those assets to maximize its value. This is done through the orchestration of people, processes, technologies, and policies within an enterprise.
- Data governance aims to gain more value from data. It lays the foundation for enterprises to achieve their digital strategies. It is a management system consisting of organizations, regulations, processes, and tools.

Benefits of Data Governance



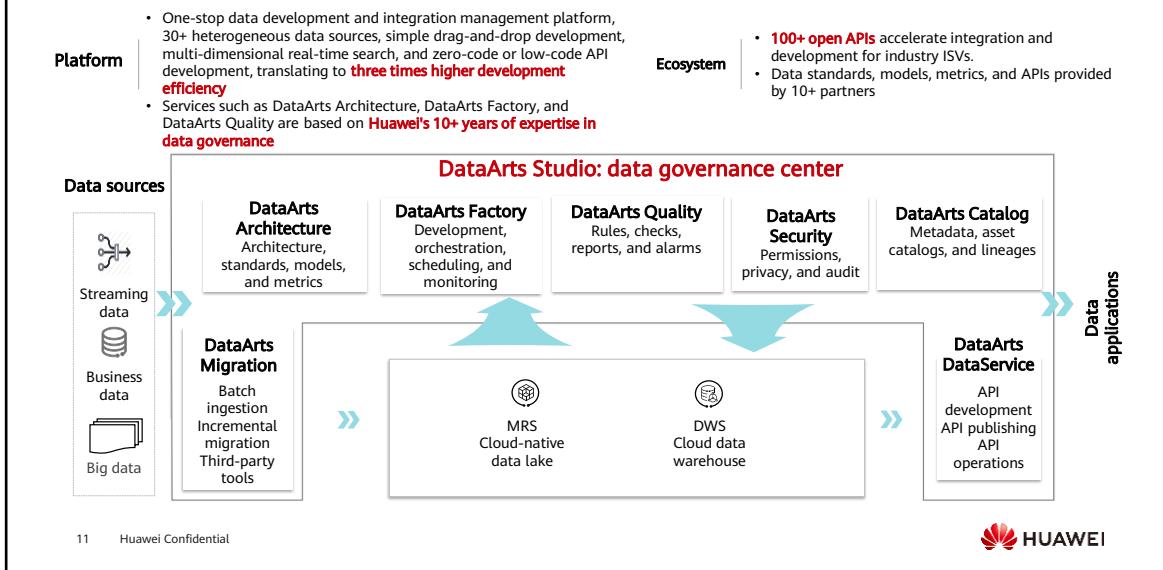
Contents

1. Data Governance
2. Huawei DataArts Studio

Huawei DataArts Studio

- DataArts Studio is a one-stop data governance and operations platform that provides intelligent data lifecycle management to help enterprises achieve digital transformation.
- It offers numerous functions, such as integrating and developing data, designing data standards, controlling data quality, managing data assets, creating data services, and ensuring data security. It supports the intelligent construction of industrial knowledge libraries and incorporates data foundations such as big data storage, computing, and analytical engines, helping enterprises easily build end-to-end intelligent data systems. These systems eliminate data silos, unify data standards, accelerate data monetization, and promote digital transformation.

DataArts Studio: One-Stop Data Development and Governance



- DataArts Studio is based on the data lake foundation 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 foundation. It can also connect to traditional enterprise data warehouses, such as Greenplum.
- Management Center
- Management Center supports data connection management and connects to the data lake foundation for activities such as data development and data governance.
- Data Integration-CDM
- CDM 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.
- Data Integration-DIS
- Data Ingestion Service (DIS) helps you transmit data to the cloud. It builds data intake streams for custom applications capable of processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, social media feeds, website click-streams, and log event location.

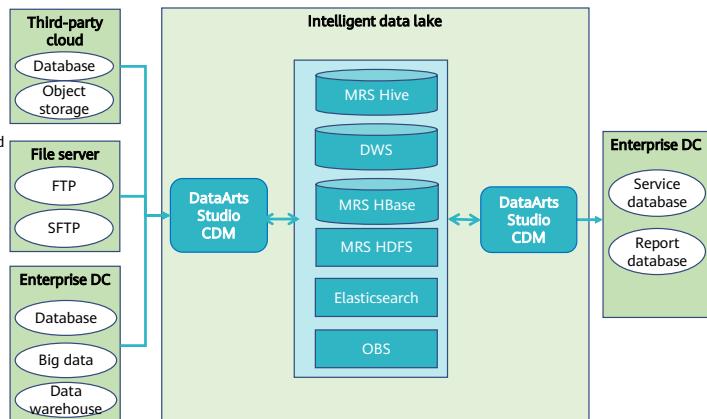
DataArts Migration: Efficient Ingestion of Multiple Heterogeneous Data Sources

- CDM

- CDM enables batch data migration between 20+ homogeneous and 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.
- CDM uses a distributed compute framework and concurrent processing techniques to help you migrate data in batches without any downtime and rapidly build desired data structures.

- DIS

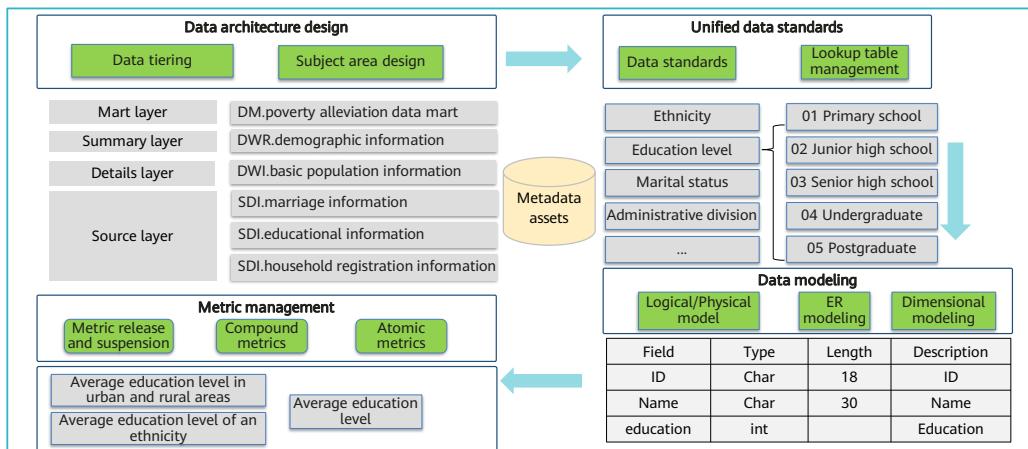
- DIS helps you transmit data to the cloud. It builds data intake streams for custom applications capable of processing or analyzing streaming data. DIS continuously captures, transmits, and stores terabytes of data from hundreds of thousands of sources every hour, such as logs, social media feeds, website click-streams, and log event location.



DataArts Architecture: Visualized, Automated, and Intelligent Data Modeling

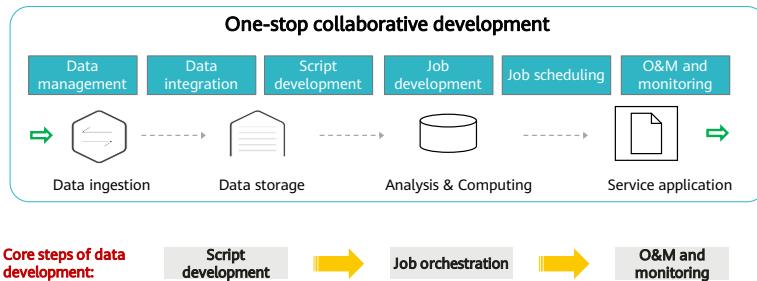
- DataArts Architecture incorporates data governance methods. You can use it to visualize data governance operations, connect data from different layers, formulate data standards, and generate data assets. You can standardize your data through ER modeling and dimensional modeling. Data Design is a good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

DataArts Architecture: Visualized, Automated, and Intelligent Data Modeling



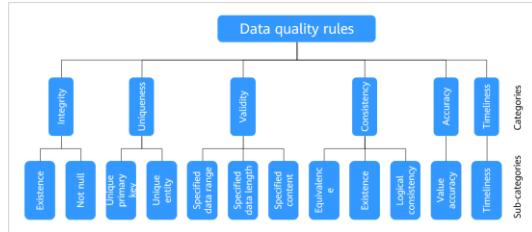
DataArts Factory: One-stop Collaborative Development

- The DataArts Factory module of DataArts Studio 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.



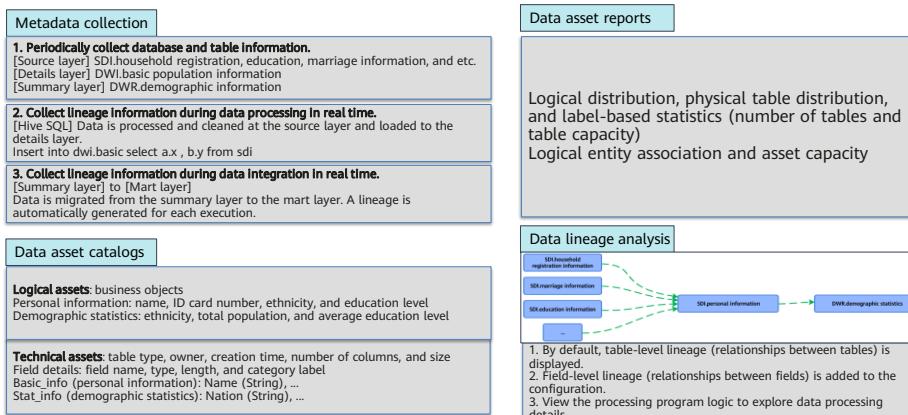
DataArts Quality: Verifiable and Controllable

- DataArts Quality can monitor your metrics and data quality, and screen out unqualified data in a timely manner.
- Business metric monitoring
 - You can use DataArts Quality to monitor the quality of data in your databases. You can create metrics, rules, or scenarios that meet your requirements and schedule them in real time or recursively.
- Data quality monitoring
 - You can create data quality rules to check whether the data in your databases is accurate in real time.
 - Qualified data must meet the following requirements: integrity, validity, timeliness, consistency, accuracy, and uniqueness. You can standardize data and periodically monitor data across columns, rows, and tables based on quality rules.



DataArts Catalog: End-to-End Data Asset Visualization

- 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.

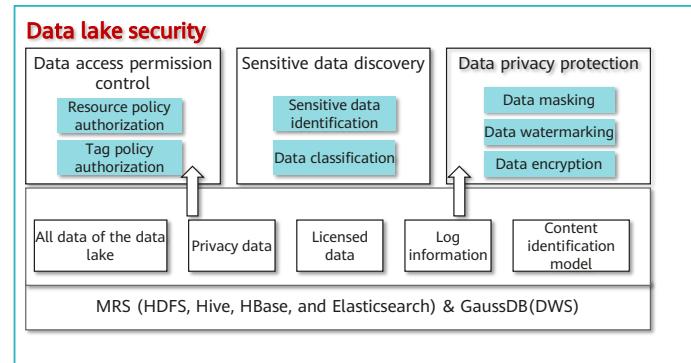


DataArts DataService: Improved Access, Query, and Search Efficiency

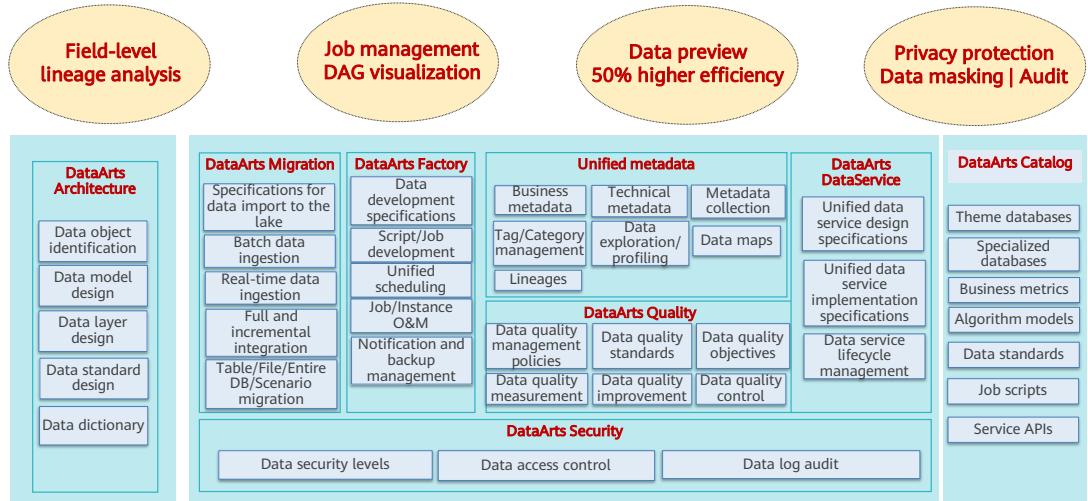
- DataArts DataService enables you to manage your enterprise APIs centrally, and controls access to your subjects, profiles, and metrics. It helps improve the experience for data consumers and the efficiency of data asset monetization. You can use DataArts DataService to generate APIs and register APIs with DataArts DataService for unified management and publication.
- DataArts DataService uses a serverless architecture. You only need to focus on the API query logic, without worrying about infrastructure such as the runtime environment. DataArts DataService supports the elastic scaling of compute resources, significantly reducing O&M costs.

DataArts Security: All-Round Protection

- Cyber 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 you 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.
- Data security
 - DataArts Studio provides a review mechanism for key processes in DataArts Architecture and DataArts DataService.
 - Data is managed by level and category throughout the lifecycle, ensuring data privacy compliance and traceability.



The Advantages of DataArts Studio



21 Huawei Confidential



- DataArts Studio competitiveness: one-stop data governance, job management, security management, and data services

Summary

- This chapter introduced data governance and DataArts Studio (Huawei's data governance service).

Quiz

1. What is the purpose of data governance?
2. What functions does DataArts Studio provide?

- Answers:

- 1. Data is a core asset of enterprises. Enterprises need to establish data dictionaries to better manage their increasingly important data. In addition, a mechanism is badly needed to continuously improve data quality. If data value is properly mined and data risks are effectively controlled, enterprises can simplify their management, integrate their service flows, improve operational efficiency, and have faithful displays of their operational results. To make scientific decisions, we must have correct data. Unified data architectures and standards are essential preconditions for efficient operations and clear and effective communications between enterprise departments. Currently, enterprises are confronted with many problems. They lack unified data standards. Their service systems are siloed. Their core data cannot be correctly identified and streamlined. To effectively manage enterprise data, maximize data value, and lay a solid foundation for digital transformation, enterprises are in urgent need of a comprehensive, well developed system of data governance.
- 2. DataArts Migration, DataArts Architecture, DataArts Factory, DataArts Quality, DataArts Catalog, DataArts DataService, and DataArts Security

Acronyms and Abbreviations

- MPPDB: massively parallel processing database, a database for large-scale parallel processing
- ETL: extract, transform, and load, a three-phase process where data is extracted from a source, transformed, and loaded to a destination
- RDS: Relational Database Service
- DWS: Data Warehouse Service
- CDM: Cloud Data Migration
- OBS: Object Storage Service
- FTP: File Transfer Protocol
- SFTP: SSH File Transfer Protocol
- DAG: Direct Acyclic Graph

Recommendations

- Official Huawei Cloud website
 - <https://www.huaweicloud.com/intl/en-us/>
- Huawei Talent
 - <https://e.huawei.com/en/talent/portal/#/>
- Huawei technical support
 - <https://support.huawei.com/enterprise/en/index.html>

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive
statements including, without limitation, statements regarding
the future financial and operating results, future product
portfolio, new technology, etc. There are a number of factors that
could cause actual results and developments to differ materially
from those expressed or implied in the predictive statements.
Therefore, such information is provided for reference purpose
only and constitutes neither an offer nor an acceptance. Huawei
may change the information at any time without notice.

